



Apple® IIe Technical Reference Manual



Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts Menlo Park, California
Don Mills, Ontario Wokingham, England Amsterdam
Sydney Singapore Tokyo Mexico City Bogotá
Santiago San Juan

Copyright © 1985 by Apple Computer, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

ISBN 0-201-17720-X

ABCDEFGHIJ-DO-898765

First printing, July 1985



Apple® IIe Technical Reference Manual

Contents

	List of Figures and Tables	xviii
	Radio and Television Interference	xxv
	<hr/>	
PREFACE	About This Manual	xxvii
	Contents of This Manual	xxvii
	The Enhanced Apple IIe	xxix
	Physical Changes	xxix
	Startup Drives	xxix
	Video Firmware	xxx
	Video Enhancements	xxx
	Applesoft 80-Column Support	xxx
	Applesoft Lowercase Support	xxxi
	Apple II Pascal	xxxi
	System Monitor Enhancements	xxxi
	Interrupt Handling	xxxi
	Symbols Used in This Manual	xxxii
	<hr/>	
CHAPTER 1	Introduction	1
	Removing the Cover	2
	The Keyboard	3
	The Speaker	3
	The Power Supply	4
	The Circuit Board	4
	Connectors on the Circuit Board	6
	Connectors on the Back Panel	8

The Keyboard	10
Reading the Keyboard	12
The Video Display Generator	16
Text Modes	18
Text Character Sets	19
40-Column Versus 80-Column Text	20
Graphics Modes	22
Low-Resolution Graphics	22
High-Resolution Graphics	23
Double-High-Resolution Graphics	25
Video Display Pages	26
Display Mode Switching	28
Addressing Display Pages Directly	30
Secondary Inputs and Outputs	37
The Speaker	37
Cassette Input and Output	38
The Hand Control Connector Signals	39
Annunciator Outputs	40
Strobe Output	40
Switch Inputs	41
Analog Inputs	42
Summary of Secondary I/O Locations	42

Using the I/O Subroutines	47
Apple II Compatibility	48
The 80-Column Firmware	49
The Old Monitor	50
The Standard I/O Links	50
Standard Output Features	51
COUT Output Subroutine	51
Control Characters With COUT1 and BASICOUT	52
The Stop-List Feature	54
The Text Window	54
Inverse and Flashing Text	56
Standard Input Features	57
RDKEY Input Subroutine	57
KEYIN Input Subroutine	58
Escape Codes	58
Cursor Motion in Escape Mode	58
GETLN Input Subroutine	60
Editing With GETLN	61
Cancel Line	61
Backspace	61
Retype	62
Monitor Firmware Support	62
BASICOUT	63
CLREOL	63
CLEOLZ	64
CLREOP	64
CLRSCR	64

CLRTOP	64
COUT	64
COUT1	64
CROUT	64
CROUT1	65
HLINE	65
HOME	65
PLOT	65
PRBL2	65
PRBYTE	65
PRERR	65
PRHEX	66
PRNTAX	66
SCRN	66
SETCOL	66
VTABZ	66
VLINE	66
I/O Firmware Support	67
PINIT	67
PREAD	67
PWRITE	68
PSTATUS	69

Main Memory Map	72
RAM Memory Allocation	74
Reserved Memory Pages	75
Page Zero	75
The 65C02 Stack	75
The Input Buffer	76
Link-Address Storage	76
The Display Buffers	76
Bank-Switched Memory	79
Setting Bank Switches	80
Reading Bank Switches	83
Auxiliary Memory and Firmware	84
Memory Mode Switching	86
Auxiliary-Memory Subroutines	88
Moving Data to Auxiliary Memory	89
Transferring Control to Auxiliary Memory	90
The Reset Routine	91
The Cold-Start Procedure	92
The Warm-Start Procedure	92
Forced Cold Start	93
The Reset Vector	93
Automatic Self-Test	95

Invoking the Monitor	98
Syntax of Monitor Commands	99
Monitor Memory Commands	100
Examining Memory Contents	100
Memory Dump	100
Changing Memory Contents	103
Changing One Byte	103
Changing Consecutive Locations	104
ASCII Input Mode	104
Moving Data in Memory	105
Comparing Data in Memory	107
Searching for Bytes in Memory	108
Examining and Changing Registers	108
Monitor Cassette Tape Commands	109
Saving Data on Tape	109
Reading Data From Tape	110
Miscellaneous Monitor Commands	112
Inverse and Normal Display	112
Back to BASIC	112
Redirecting Input and Output	113
Hexadecimal Arithmetic	114
Special Tricks With the Monitor	114
Multiple Commands	114
Filling Memory	115
Repeating Commands	116
Creating Your Own Commands	117

Machine-Language Programs	118
Running a Program	118
Disassembled Programs	119
The Mini-Assembler	121
Starting the Mini-Assembler	121
Restrictions	121
Using the Mini-Assembler	122
Mini-Assembler Instruction Formats	124
Summary of Monitor Commands	125
Examining Memory	125
Changing the Contents of Memory	126
Moving and Comparing	126
The Examine Command	126
The Search Command	126
Cassette Tape Commands	126
Miscellaneous Monitor Commands	127
Running and Listing Programs	127
The Mini-Assembler	128

CHAPTER 6

Programming for Peripheral Cards	129
Peripheral-Card Memory Spaces	130
Peripheral-Card I/O Space	130
Peripheral-Card ROM Space	131
Expansion ROM Space	132
Peripheral-Card RAM Space	134

I/O Programming Suggestions	135
Finding the Slot Number With ROM Switched In	136
I/O Addressing	136
RAM Addressing	138
Changing the Standard I/O Links	139
Other Uses of I/O Memory Space	140
Switching I/O Memory	141
Developing Cards for Slot 3	143
Pascal 1.1 Firmware Protocol	144
Device Identification	144
I/O Routine Entry Points	145
Interrupts on the Enhanced Apple IIe	146
What Is an Interrupt?	147
Interrupts on Apple IIe Series Computers	148
Rules of the Interrupt Handler	149
Interrupt Handling on the 65C02 and 6502	150
The Interrupt Vector at \$FFFE	150
The Built-in Interrupt Handler	151
Saving the Apple IIe's Memory Configuration	152
Managing Main and Auxiliary Stacks	152
The User's Interrupt Handler at \$3FE	154
Handling Break Instructions	155
Interrupt Differences: Apple IIe Versus Apple IIc	156

Environmental Specifications	158
The Power Supply	159
The Power Connector	161
The 65C02 Microprocessor	161
65C02 Timing	162
The Custom Integrated Circuits	164
The Memory Management Unit	164
The Input/Output Unit	166
The PAL Device	168
Memory Addressing	168
ROM Addressing	169
RAM Addressing	170
Dynamic-RAM Refreshment	170
Dynamic-RAM Timing	171
The Video Display	173
The Video Counters	174
Display Memory Addressing	175
Display Address Mapping	176
Video Display Modes	179
Text Displays	179
Low-Resolution Display	182
High-Resolution Display	183
Double-High-Resolution Display	185
Video Output Signals	186

Built-in I/O Circuits	187
The Keyboard	187
Connecting a Keypad	188
Cassette I/O	189
The Speaker	189
Game I/O Signals	190
Expanding the Apple IIe	192
The Expansion Slots	192
The Peripheral Address Bus	192
The Peripheral Data Bus	193
Loading and Driving Rules	193
Interrupt and DMA Daisy Chains	193
Auxiliary Slot	197
80-Column Display Signals	197

APPENDIX A

The 65C02 Microprocessor	205
Differences Between 6502 and 65C02	206
Different Cycle Times	206
Different Instruction Results	207
Data Sheet	207

APPENDIX B

Directory of Built-in Subroutines	217
-----------------------------------	-----

APPENDIX C

Apple II Family Differences 225

Keyboard 226
Apple Keys 226
Character Sets 226
80-Column Display 227
Escape Codes and Control Characters 227
Built-in Language Card 227
Auxiliary Memory 228
Auxiliary Slot 228
Back Panel and Connectors 228
Soft Switches 228
Built-in Self-Test 229
Forced Reset 229
Interrupt Handling 229
Vertical Sync for Animators 229
Signature Byte 230
Hardware Implementation 230

APPENDIX D

Operating Systems and Languages 231

Operating Systems 232
 ProDOS 232
 DOS 3.3 232
 Pascal Operating System 232
 CP/M 233

Languages	233
Assembly Language	233
Applesoft BASIC	233
Interger BASIC	233
Pascal Language	234
FORTRAN	234

APPENDIX E	Conversion Tables	235
	Bits and Bytes	236
	Hexadecimal and Decimal	238
	Hexadecimal and Negative Decimal	240
	Graphics Bits and Pieces	242
	Eight-Bit Code Conversions	244

APPENDIX F	Frequently Used Tables	253
------------	------------------------	-----

APPENDIX G	Using an 80-Column Text Card	267
	Starting Up With Pascal or CP/M	268
	Starting Up With ProDOS or DOS 3.3	269
	Using the GET Command	269
	When to Switch Modes Versus When to Deactivate	270
	Display Features With the Text Card	270
	INVERSE, FLASH, NORMAL, HOME	270

Tabbing With the Original Apple IIe	271
Comma Tabbing With the Original Apple IIe	271
HTAB and POKE 1403	272
Using Control-Characters With the Card	272
Control Characters and Their Functions	273
How to Use Control-Character Codes in Programs	274
A Word of Caution to Pascal Programmers	275

APPENDIX H

Programming With the Super Serial Card	277
Locating the Card	278
Operating Modes	278
Operating Commands	279
The Command Character	280
Baud Rate, nB	280
Data Format, nD	281
Parity, nP	281
Set Time Delay, nC, nL, and nF	282
Echo Characters to the Screen, E__E/D	283
Automatic Carriage Return, C	283
Automatic Line Feed, LE/D	284
Mask Line Feed In, M__E/D	284
Reset Card, R	284
Specify Screen Slot, S	284
Translate Lowercase Characters, nT	284
Suppress Control Characters, Z	285
Find Keyboard, F__E/D	285
XOFF Recognition, X__E/D	286
Tab in BASIC, T__E/D	286

Terminal Mode 286

 Entering Terminal Mode, T 286

 Transmitting a Break, B 287

 Special Characters, S_E/D 287

 Quitting Terminal Mode, Q 287

SSC Error Codes 287

The ACIA 289

SSC Firmware Memory Use 289

 Zero-Page Location 290

 Peripheral Card I/O Space 290

 Scratchpad RAM Location 292

APPENDIX I

Monitor ROM Listing	293
---------------------	-----

Glossary	377
Bibliography	399
Index	401
Tell Apple Card	

Figures and Tables

CHAPTER 1

Introduction

1

Figure 1-1	Removing the Cover	2
Figure 1-2	The Apple IIe With the Cover Off	2
Figure 1-3	The Apple IIe Keyboard	3
Figure 1-4	The Circuit Board	5
Figure 1-5	The Expansion Slots	7
Figure 1-6	The Auxiliary Slot	7
Figure 1-7	The Back Panel Connectors	8

CHAPTER 2

Built-in I/O Devices

9

Figure 2-1	The Keyboard	11
Table 2-1	Apple IIe Keyboard Specifications	11
Table 2-2	Keyboard Memory Locations	12
Table 2-3	Keys and ASCII Codes	14
Table 2-4	Video Display Specifications	17
Table 2-5	Display Character Sets	20
Figure 2-2	40-Column Text Display	21
Figure 2-3	80-Column Text Display	21
Table 2-6	Low-Resolution Graphics Colors	23
Figure 2-4	High-Resolution Display Bits	24
Table 2-7	High-Resolution Graphics Colors	25
Table 2-8	Double-High-Resolution Graphics Colors	26
Table 2-9	Video Display Page Locations	28
Table 2-10	Display Soft Switches	29
Figure 2-5	Map of 40-Column Text Display	32
Figure 2-6	Map of 80-Column Text Display	33
Figure 2-7	Map of Low-Resolution Graphics Display	34

Figure 2-8	Map of High-Resolution Graphics Display	35
Figure 2-9	Map of Double-High-Resolution Graphics Display	36
Table 2-11	Annunciator Memory Locations	40
Table 2-12	Secondary I/O Memory Location	43

CHAPTER 3

Built-in I/O Firmware 45

Table 3-1	Monitor Firmware Routines	46
Table 3-2	Apple II Mode	48
Table 3-3a	Control Characters With 80-Column Firmware Off	52
Table 3-3b	Control Characters With 80-Column Firmware On	52
Table 3-4	Text Window Memory Locations	55
Table 3-5	Text Format Control Values	56
Table 3-6	Escape Codes	59
Table 3-7	Prompt Characters	60
Table 3-8	Video Firmware Routines	62
Table 3-9	Port 3 Firmware Protocol Table	67
Table 3-10	Pascal Video Control Functions	68

CHAPTER 4

Memory Organization 71

Figure 4-1	System Memory Map	73
Figure 4-2	RAM Allocation Map	74
Table 4-1	Monitor Zero-Page Use	77
Table 4-2	Applesoft Zero-Page Use	77
Table 4-3	Integer BASIC Zero-Page Use	78
Table 4-4	DOS 3.3 Zero-Page Use	78
Table 4-5	ProDOS MLI and Disk-Driver Zero-Page Use	79

Figure 4-3	Bank-Switched Memory Map	80
Table 4-6	Bank Select Switches	82
Figure 4-4	Memory Map With Auxiliary Memory	85
Table 4-7	Auxiliary-Memory Select Switches	87
Table 4-8	48K RAM Transfer Routines	88
Table 4-9	Parameters for AUXMOVE Routine	89
Table 4-10	Parameters for XFER Routine	90
Table 4-11	Page 3 Vectors	94

CHAPTER 5

Using the Monitor 97

Table 5-1	Mini-Assembler Address Formats	124
-----------	--------------------------------	-----

CHAPTER 6

Programming for Peripheral Cards 129

Table 6-1	Peripheral-Card I/O Memory Locations Enabled by DEVICE SELECT'	131
Table 6-2	Peripheral-Card ROM Memory Locations Enabled by I/O SELECT'	132
Figure 6-1	Expansion ROM Enable Circuit	133
Figure 6-2	ROM Disable Address Decoding	133
Table 6-3	Peripheral-Card RAM Memory Locations	134
Table 6-4	Peripheral-Card I/O Base Addresses	137
Figure 6-3	I/O Memory Map	141
Table 6-5	I/O Memory Switches	142
Table 6-6	Peripheral-Card Device-Class Assignments	144
Table 6-7	I/O Routine Offsets and Registers Under Pascal 1.1 Protocol	146

Figure 6-4	Interrupt Handling Sequence 151
Table 6-8	BRK Handler Information 155
Table 6-9	Memory Configuration Information 155

CHAPTER 7

Hardware Implementation		157
Table 7-1	Summary of Environmental Specifications 158	
Table 7-2	Power Supply Specifications 159	
Table 7-3	Power Connector Signal Specifications 161	
Table 7-4	65C02 Microprocessor Specifications 162	
Table 7-5	65C02 Timing Signal Descriptions 163	
Figure 7-1	65C02 Timing Signals 163	
Figure 7-2	The MMU Pinouts 165	
Table 7-6	The MMU Signal Descriptions 165	
Figure 7-3	The IOU Pinouts 167	
Table 7-7	The IOU Signal Descriptions 167	
Figure 7-4	The PAL Pinouts 168	
Table 7-8	The PAL Signal Descriptions 168	
Figure 7-5	The 2364 ROM Pinouts 169	
Figure 7-6	The 2316 ROM Pinouts 169	
Figure 7-7	The 2333 ROM Pinouts 169	
Figure 7-8	The 64K RAM Pinouts 170	
Table 7-9	RAM Address Multiplexing 171	
Figure 7-9	RAM Timing Signals 172	
Table 7-10	RAM Timing Signal Descriptions 173	
Table 7-11	Display Address Transformation 176	
Figure 7-10	40-Column Text Display Memory 177	
Table 7-12	Display Memory Addressing 178	

Table 7-13	Memory Address Bits for Display Modes	178
Figure 7-11a	7 MHz Video Timing Signals	180
Figure 7-11b	14 MHz Video Timing Signals	181
Table 7-14	Character-Generator Control Signals	182
Table 7-15	Internal Video Connector Signals	186
Table 7-16	Keyboard Connector Signals	188
Table 7-17	Keypad Connector Signals	188
Table 7-18	Speaker Connector Signals	189
Table 7-19	Game I/O Connector Signals	191
Figure 7-12	Peripheral-Signal Timing	194
Table 7-20	Expansion Slot Signals	195
Table 7-21	Auxiliary Slot Signals	198
Figure 7-13	Schematic Diagram	200

APPENDIX A

The 65C02 Microprocessor		205
Table A-1	Cycle Time Differences	206

APPENDIX E

Conversion Tables		235
Table E-1	What a Bit Can Represent	236
Figure E-1	Bits, Nibbles, and Bytes	237
Table E-2	Hexadecimal/Decimal Conversion	238
Table E-3	Hexadecimal to Negative Decimal Conversion	240
Table E-4	Hexadecimal Values for High-Resolution Dot Patterns	242
Table E-5	Control Characters, High Bit Off	245
Table E-6	Special Characters, High Bit Off	246

Table E-7	Uppercase Characters, High Bit Off	247
Table E-8	Lowercase Characters, High Bit Off	248
Table E-9	Control Characters, High Bit On	249
Table E-10	Special Characters, High Bit On	250
Table E-11	Uppercase Characters, High Bit On	251
Table E-12	Lowercase Characters, High Bit On	252

APPENDIX F

Frequently Used Tables		253
Table 2-3	Keys and ASCII Codes	254
Table 2-2	Keyboard Memory Location	255
Table 2-4	Video Display Specifications	256
Table 2-8	Double-High-Resolution Graphics Colors	257
Table 2-9	Video Display Page Locations	257
Table 2-10	Display Soft Switches	258
Table 3-1	Monitor Firmware Routines	259
Table 3-3a	Control Characters With 80-Column Firmware Off	260
Table 3-3b	Control Characters With 80-Column Firmware On	260
Table 3-5	Text Format Control Values	261
Table 3-6	Escape Codes	262
Table 3-10	Pascal Video Control Functions	263
Table 4-6	Bank Select Switches	264
Table 4-7	Auxiliary-Memory Select Switches	265
Table 4-8	48K RAM Transfer Routines	265
Table 6-5	I/O Memory Switches	266
Table 6-6	I/O Routine Offsets and Registers Under Pascal 1.1 Protocol	266

APPENDIX G

	Using an 80-Column Text Card	267
Table G-1	Control Characters With 80-Column Firmware On	273

APPENDIX H

	Programming With the Super Serial Card	277
Table H-1	Baud Rate Selections	280
Table H-2	Data Format Selections	281
Table H-3	Parity Selections	281
Table H-4	Time Delay Selections	282
Table H-5	Lowercase Character Display Options	285
Table H-6	STSBYTE Bit Definitions	287
Table H-7	Error Codes and Bits	288
Table H-8	Memory Use Map	289
Table H-9	Zero-Page Locations Used by the SSC	290
Table H-10	Address Register Bits Interpretation	291
Table H-11	Scratchpad RAM Locations Used by the SSC	292

Radio and Television Interference

The equipment described in this manual generates and uses radio-frequency energy. If it is not installed and used properly—that is, in strict accordance with our instructions—it may cause interference with radio and television reception.

This equipment has been tested and complies with the limits for a Class B computing device in accordance with the specifications in Subpart J, Part 15, of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation, especially if a “rabbit ear” television antenna is used. (A “rabbit ear” antenna is the telescoping-rod type usually contained on television receivers.)

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripherals. To further isolate the problem, disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it was caused by either the peripheral device or the I/O cable. These devices usually require shielded I/O cables. For Apple peripherals, you can obtain the proper **shielded cable** from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

A **shielded cable** is a cable that uses a metallic wrap around the wires to reduce the potential effects of radio frequency interference.

If your computer does cause interference to radio or television reception, you can try to correct the interference by using one or more of the following measures:

- ☐ Turn the television or radio antenna until the interference stops.
- ☐ Move the computer to one side or the other of the television or radio.
- ☐ Move the computer farther away from the television or radio.

- Plug the computer into an outlet that is on a different circuit than the television or radio. (That is, make certain the computer and the radio or television set are on circuits controlled by different circuit breakers or fuses.)
- Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and television.

If necessary, you should consult your Apple-authorized dealer or an experienced radio/television technician for additional suggestions.

This is the reference manual for the Apple IIe personal computer. It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIe and provides the technical information that peripheral-card designers and programmers need.

This manual contains a lot of information about the way the Apple IIe works, but it doesn't tell you how to use the Apple IIe. For this, you should read the other Apple IIe manuals, especially the following:

- *Apple IIe Owner's Manual*
- *The Applesoft Tutorial*

Contents of This Manual

The material in this manual is presented roughly in order of increasing intimacy with the hardware; the farther you go in the manual, the more technical the material becomes. The main subject areas are

- introduction: Preface and Chapter 1
- use of built-in features: Chapters 2 and 3
- how the memory is organized: Chapter 4
- information for programmers: Chapters 5 and 6
- hardware implementation: Chapter 7
- additional information: appendixes, glossary, and bibliography.

Chapter 1 identifies the main parts of the Apple IIe and tells where in the manual each part is described.

The next two chapters describe the built-in input and output features of the Apple IIe. This part of the manual includes information you need for low-level programming on the Apple IIe. Chapter 2 describes the built-in I/O features and Chapter 3 tells you how to use the firmware that supports them.

Chapter 4 describes the way the Apple II's memory space is organized, including the allocation of programmable memory for the video display buffers.

Chapter 5 is a user manual for the Monitor that is included in the built-in firmware. The Monitor is a system program that you can use for program debugging at the machine level.

Chapter 6 describes the programmable features of the peripheral-card connectors and gives guidelines for their use. It also describes interrupt programming on the Apple IIe.

Chapter 7 is a description of the hardware that implements the features described in the earlier chapters. This information is included primarily for programmers and peripheral-card designers, but it will also help you if you just want to understand more about the way the Apple IIe works.

Additional reference information appears in the appendixes. Appendix A is the manufacturer's description of the Apple IIe's microprocessor.

Appendix B is a directory of the built-in I/O subroutines, including their functions and starting addresses.

Appendix C describes differences among Apple II family members.

Appendix D describes some of the operating systems and languages supported by Apple Computer for the Apple IIe.

Appendix E contains conversion tables of interest to programmers.

Appendix F contains additional copies of some of the tables that appear in the body of the manual. The ones you will need to refer to often are duplicated here for easy reference.

Appendix G contains information about using Apple IIe 80-column text cards with the Apple IIe and high level languages.

Appendix H discusses programming on the Apple IIe with the Apple Super Serial Card.

Appendix I contains the source listing of the Monitor firmware. You can refer to it to find out more about the operation of the Monitor subroutines listed in Appendix B.

Following Appendix I is a glossary defining many of the technical terms used in this manual. Some terms that describe the use of the Apple IIe are defined in the glossaries of the other manuals listed earlier.

Following the glossary, there is a selected bibliography of sources of additional information.

The Enhanced Apple IIe

Changes have been made in the Apple IIe since the original version was introduced. The new version is called the enhanced Apple IIe and is described in this manual. Where there are differences in the original Apple IIe compared with the enhanced Apple IIe, they will be called out in the manual. Otherwise, the two machines operate identically.

You can tell whether you have an original or enhanced Apple IIe when you start up the system. An original Apple IIe will display **Apple II** at the top of the monitor screen, while an enhanced Apple IIe will display **Apple IIe**.

The changes embodied in the enhanced Apple IIe are described in the following sections of this preface.

Physical Changes

The enhanced Apple IIe includes the following changes from the original Apple IIe:

- The 65C02 microprocessor, which is a new version of the 6502 microprocessor found in the original Apple IIe. The 65C02 uses less power, has 27 new **opcodes**, and runs at the same speed as the 6502. (See Chapter 7 and Appendix A.)
- A new video ROM containing the same MouseText characters found in the Apple IIc. (See Chapter 2.)
- New Monitor ROMs (the CD and EF ROMs) containing the enhanced Apple IIe firmware. (See Chapter 5.)
- The identification byte at \$FBC0 has been changed. In the original Apple IIe it was \$EA (decimal 234), in the enhanced Apple IIe it is \$E0 (decimal 224).

Opcode is short for *operation code* and is used to describe the basic instructions performed by the central processing unit of a computer.

Startup Drives

You can use startup (boot) devices other than a Disk II to start up ProDOS on the enhanced Apple IIe.

Apple II Pascal versions 1.3 and later may start up from slots 4, 5, or 6 on a Disk II, ProFile, or other Apple II disk drive. Apple II Pascal versions 1.0 through 1.2 must start up from a Disk II in slot 6.

DOS 3.3 may be started from a Disk II in any slot.

When you turn on your Apple IIe, it searches for a disk drive controller to start up from, beginning with slot 7 and working down toward slot 1. As soon as a disk controller card is found, the Apple IIe will try to load and execute the operating system found on the disk. If the drive is not a Disk II, then the operating system of the startup volume must be either ProDOS or Apple II Pascal (version 1.3 or later). If it is a Disk II, then the startup volume may be any Apple II operating system.

Video Firmware

The enhanced Apple IIe has improved 80-column firmware:

- ☐ The enhanced Apple IIe now supports lowercase input.
- ☐ `ESC CONTROL E` passes most control characters to the screen.
- ☐ `ESC CONTROL D` traps most control characters before they get to the screen.
- ☐ `ESC R` was removed because uppercase characters are no longer required by Applesoft.

Video Enhancements

Both 80-column Pascal and 80-column mode Applesoft output are faster than before and scrolling is smoother. 40-column Pascal performance is unchanged.

In the original Apple IIe, characters echoed to COUT1 during 80-column operation were printed in every other column; the enhanced Apple IIe firmware now prints the characters in each column.

Applesoft 80-Column Support

The following Applesoft routines now work in 80-column mode:

- ☐ HTAB
- ☐ TAB
- ☐ SPC
- ☐ Comma tabbing in PRINT statements

Applesoft Lowercase Support

Applesoft now lets you do all your programming in lowercase. When you list your programs, all Applesoft keywords and variable names automatically are in uppercase characters; literal strings and the contents of DATA and REM statements are unchanged.

Apple II Pascal

Apple II Pascal (version 1.2 and later) can now use a ProFile hard disk through the Pascal ProFile Manager.

To find out more, see the *Pascal ProFile Manager Manual*.

The Pascal 1.1 firmware no longer supports the control character that switches from 80-column to 40-column operation. This control character is no longer supported because it can put Pascal in a condition where the exact memory configuration is not known.

System Monitor Enhancements

Enhancements to the Apple IIe's built-in Monitor (described in Chapter 5 in this manual) include the following:

- ☐ lowercase input
- ☐ ASCII input mode
- ☐ Monitor Search command
- ☐ the Mini-Assembler

Interrupt Handling

Interrupt handler support in the enhanced Apple IIe firmware now handles any Apple IIe memory configuration.

Symbols Used in This Manual

Special text in this manual is set off in several different ways, as shown in these examples.

▲Warning

Important warnings appear in red like this. These flag potential danger to the Apple IIe, its software, or you.

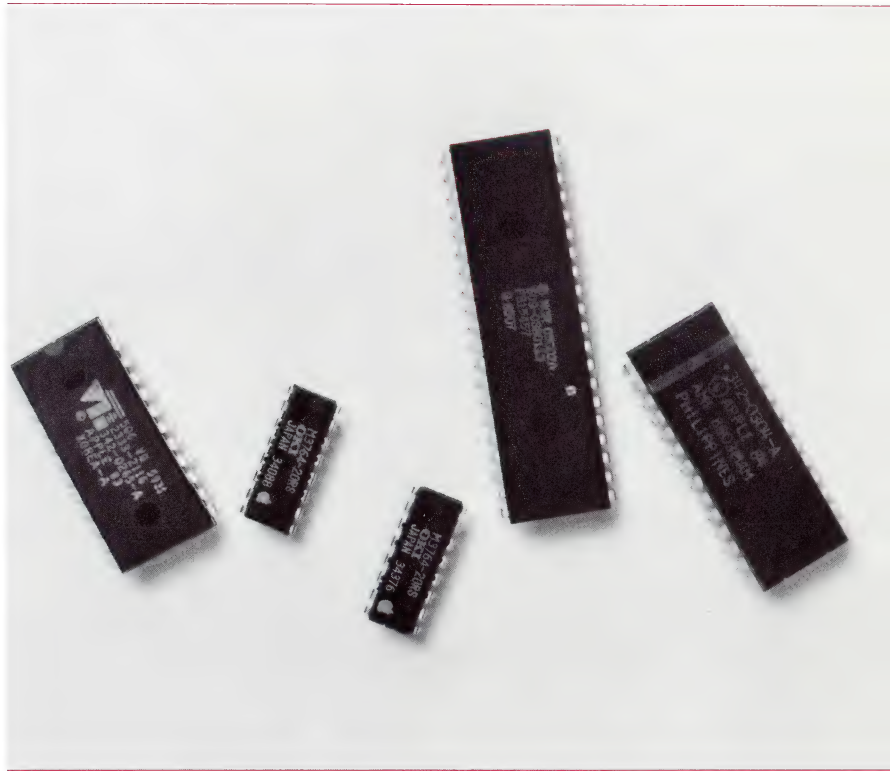
Important!

The information here is important, but non-threatening. The ways in which the original Apple IIe differs from the enhanced Apple IIe are called out this way with the tag **Original IIe** in the margin.

By the Way: Information that is useful but is incidental to the text is set off like this. You may want to skip over such information and return to it later.

Definitions, cross-references, and other short items appear in marginal glosses like this.

Terms that are defined in a marginal gloss or in the glossary appear in **boldface**.



This first chapter introduces you to the Apple IIe itself. It shows you what the inside looks like, identifies the major components that make up the machine, and tells you where to find information about each one.

Removing the Cover

Remove the cover of the Apple IIe by pulling up on the back edge until the fasteners on either side pop loose, then move the cover an inch or so toward the rear of the machine to free the front of the cover, as shown in Figure 1-1. What you will see is shown in Figure 1-2.

Figure 1-1. Removing the Cover



Figure 1-2. The Apple IIe With the Cover Off



▲Warning

There is a red LED (light-emitting diode) inside the Apple IIe, in the left rear corner of the circuit board. If the LED is on, it means that the power is on and you must turn it off before you insert or remove anything. To avoid damaging the Apple IIe, don't even *think* of changing anything inside it without first turning off the power.

ASCII stands for *American Code for Information Interchange*.

The Keyboard

The keyboard is the Apple IIe's primary input device. As shown in Figure 1-3, it has a normal typewriter layout, uppercase and lowercase, with all of the special characters in the **ASCII** character set. The keyboard is fully integrated into the machine; its operation is described in the first part of Chapter 2. Firmware subroutines for reading the keyboard are described in Chapter 3.

Figure 1-3. The Apple IIe Keyboard



The Speaker

The Apple IIe has a small loudspeaker in the bottom of the case. The speaker enables Apple IIe programs to produce a variety of sounds that make the programs more useful and interesting. The way programs control the speaker is described in Chapter 2.

The Power Supply

The power supply is inside the flat metal box along the left side of the interior of the Apple IIe. It provides power for the main board and for any peripheral cards installed in the Apple IIe.

The power supply produces four different voltages: +5V, -5V, +12V, and -12V. It is a high-efficiency switching supply; it includes special circuits that protect it and the rest of the Apple IIe against short circuits and other mishaps. Complete specifications of the Apple IIe power supply appear in Chapter 7.

▲Warning

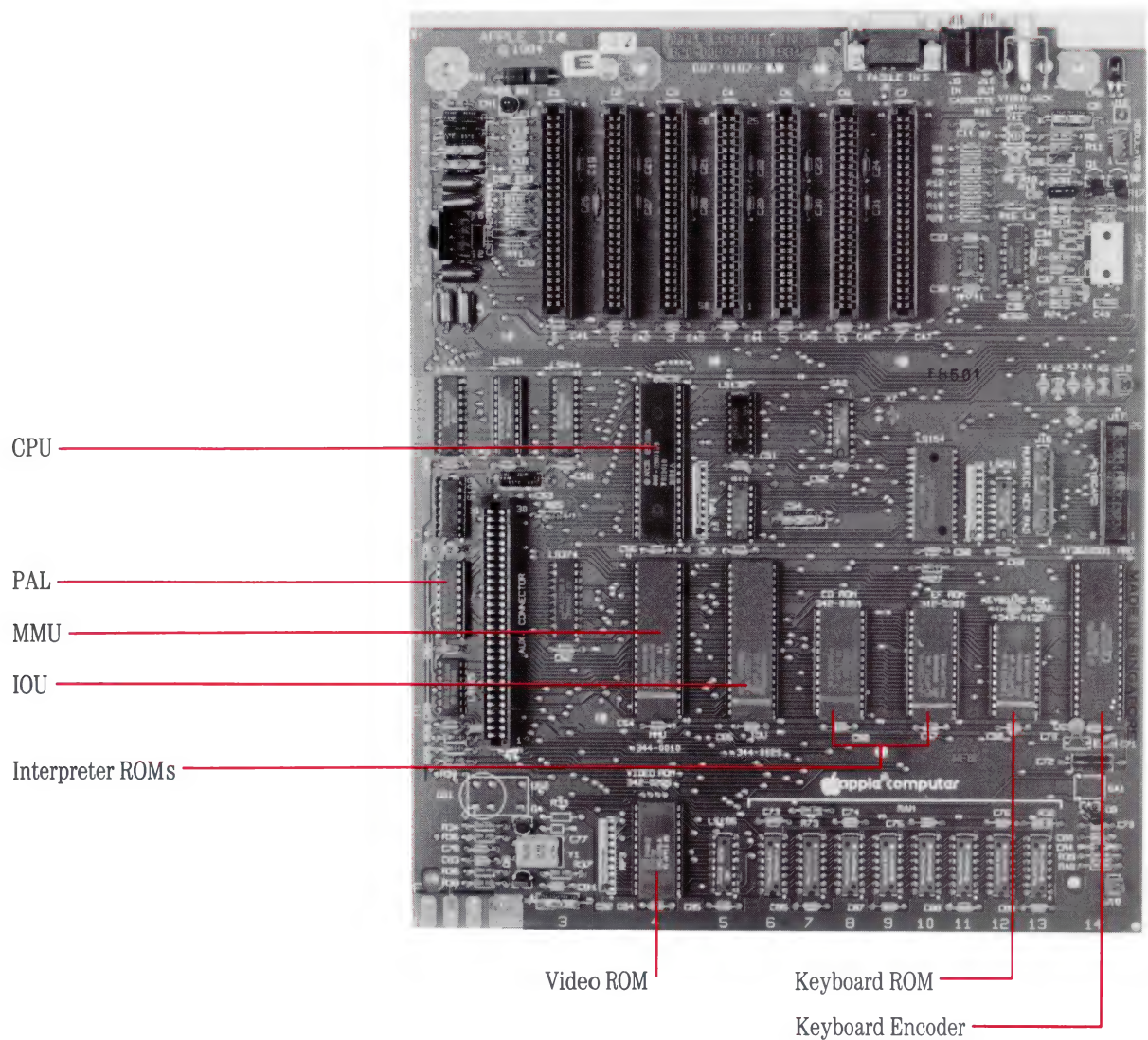
The power switch and the socket for the power cord are mounted directly on the back of the power supply's metal case. This mounting ensures that all the circuits that carry dangerous voltages are inside the power supply. Do not defeat this design feature by attempting to open the power supply.

The Circuit Board

All of the electronic parts of the Apple IIe are attached to the circuit board, which is mounted flat in the bottom of the case.

Figure 1-4 shows the main integrated circuits (ICs) in the Apple IIe. They are the central processing unit (CPU), the keyboard encoder, the keyboard read-only memory (ROM), the two interpreter ROMs, the video ROM, and the custom integrated circuits: the Input/Output Unit (IOU), the Memory Management Unit (MMU), and the Programmed Array Logic (PAL) device.

Figure 1-4. The Circuit Board



The CPU is a 65C02 microprocessor. The 65C02 is an enhanced version of the 6502, which is an eight-bit microprocessor with a sixteen-bit address bus. It uses instruction pipelining for faster processing than comparable microprocessors. In the Apple IIe, the 65C02 runs at 1.02 MHz and performs up to 500,000 eight-bit operations per second. The specifications of the 65C02 and its instruction set are given in Appendix A.

The original version of the Apple IIe uses the 6502 microprocessor. You can tell which version of Apple IIe that you have by starting up your machine. An original Apple IIe displays **Apple II** at the top of the screen during startup, while an enhanced Apple IIe displays **Apple IIe**. This manual will call out specific areas where the two versions of the Apple IIe are different.

Original IIe

The 6502 is very similar to the 65C02, but lacks 10 instructions and 2 addressing modes found on the 65C02. The 6502 is an NMOS device and so uses more power than the CMOS 65C02. Except for the differences listed above, and some minor differences in the number of clock cycles used by some instructions, the two microprocessors are identical.

The keyboard is decoded by an AY-3600-type integrated circuit and a read-only memory (ROM). These devices are described in Chapter 7.

The interpreter ROMs are integrated circuits that contain the Applesoft BASIC interpreter. The ROMs are described in Chapter 7. The Applesoft language is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.

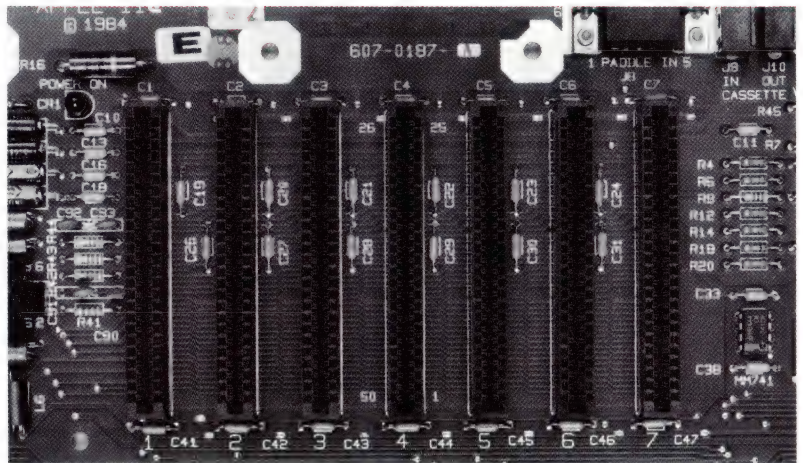
Two of the large ICs are custom-made for the Apple IIe: the MMU and the IOU. The MMU IC contains most of the logic that controls memory addressing in the Apple IIe. The organization of the memory is described in Chapter 4; the circuitry in the MMU itself is described in Chapter 7.

The IOU IC contains most of the logic that controls the built-in input/output features of the Apple IIe. These features are described in Chapter 2 and Chapter 3; the IOU circuits are described in Chapter 7.

Connectors on the Circuit Board

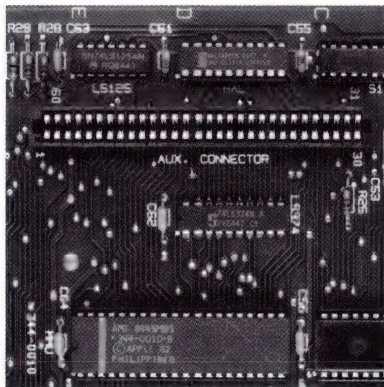
The seven slots lined up along the back of the Apple IIe circuit board are the expansion slots, sometimes called peripheral slots. (See Figure 1-5.) These slots make it possible to attach additional hardware to the Apple IIe. Chapter 6 tells you how your programs deal with the devices that plug into these slots; Chapter 7 describes the circuitry for the slots themselves.

Figure 1-5. The Expansion Slots



The large slot next to the left-hand side of the circuit board is the auxiliary slot (Figure 1-6). If your Apple IIe has an Apple IIe 80-column text card, it will be installed in this slot. The 80-column display option is fully integrated into the Apple IIe; it is described along with the other display features in Chapter 2. The hardware and firmware interfaces to this card are described in Chapter 7.

Figure 1-6. The Auxiliary Slot

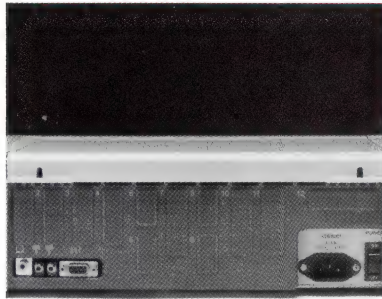


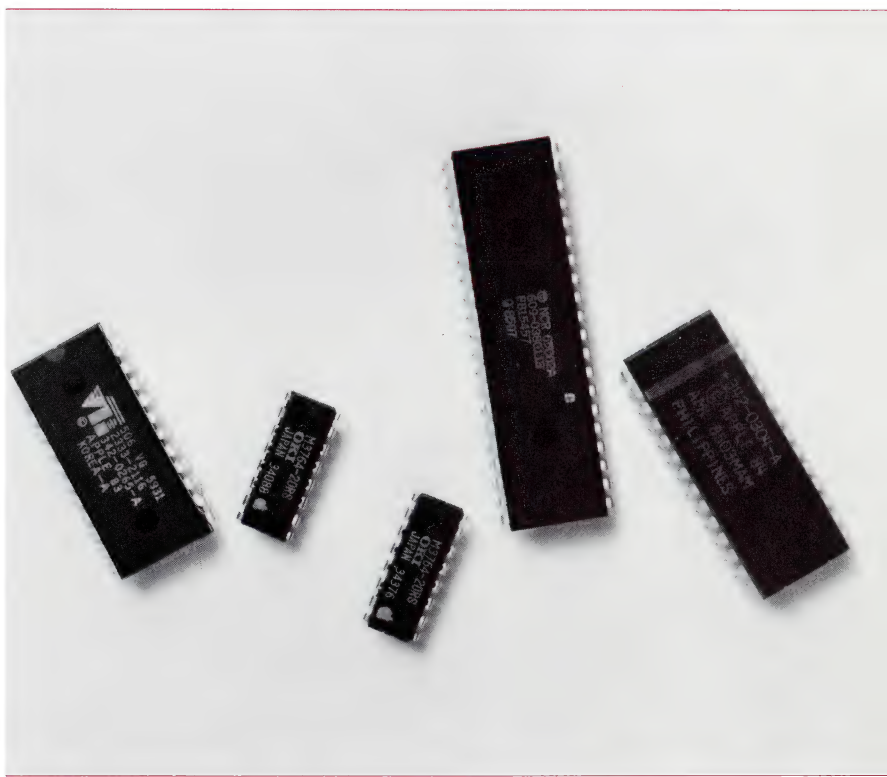
There are also smaller connectors for game I/O and for an internal RF (radio frequency) modulator. These connectors are described in Chapter 7.

Connectors on the Back Panel

The back of the Apple IIe has two miniature phone jacks for connecting a cassette recorder, an RCA-type jack for a video monitor, and a 9-pin D-type miniature connector for the hand controls, as shown in Figure 1-7. In addition to these, there are spaces for additional connectors used with the peripheral cards installed in the Apple IIe. The installation manuals for the peripheral cards contain instructions for installing the peripheral connectors.

Figure 1-7. The Back Panel Connectors





This chapter describes the input and output (I/O) devices built into the Apple IIe in terms of their functions and the way they are used by programs. The built-in I/O devices are

- the keyboard
- the video-display generator
- the speaker
- the cassette input and output
- the game input and output.

At the lowest level, programs use the built-in I/O devices by reading and writing to dedicated memory locations. This chapter lists these locations for each I/O device. It also gives the locations of the internal soft-switches that select the different display modes of the Apple IIe.

For descriptions of the built-in I/O hardware refer to Chapter 7.

Built-in I/O firmware routines are described in Chapter 3.

Built-in I/O Routines: This method of input and output—loading and storing directly to specific locations in memory—is not the only method you can use. For many of your programs, it may be more convenient to call the built-in I/O routines stored in the Apple IIe’s firmware.

The Keyboard

The primary input device of the Apple IIe is its built-in keyboard. The keyboard has 63 keys and is similar to a typewriter keyboard. The Apple IIe keyboard has automatic repeat on all keys: hold the key down to repeat. It also has N-key rollover, which means that you can hold down any number of keys while typing another. Of course, if you hold the keys down much longer than the length of time you would hold them down during normal typing, the automatic-repeat function will start repeating the last key you pressed.

The keyboard arrangement shown in Figure 2-1 is the standard one used in the United States. The specifications for the keyboard are given in Table 2-1. Apple IIe’s manufactured for sale outside the United States have a slightly different standard keyboard arrangement and include provisions for switching between two different arrangements.

Figure 2-1. The Keyboard

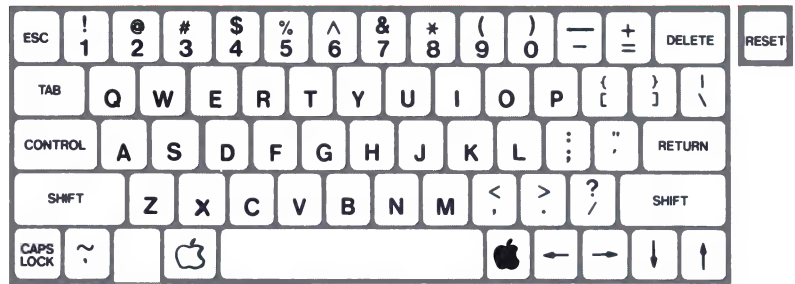










Table 2-1. Apple IIe Keyboard Specifications

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Features:	Automatic repeat, two-key rollover
Special function keys:	RESET,  , 
Cursor movement keys:	 ,  ,  ,  , RETURN, DELETE, TAB
Modifier keys:	CONTROL, SHIFT, CAPS LOCK, ESC
Electrical Interface:	AY-5-3600 keyboard encoder

In addition to the keys normally used for typing characters, there are four cursor-control keys with arrows: left, right, down, and up. The cursor-control keys can be read the same as other keys; their codes are \$08, \$15, \$0A, and \$0B. (See Table 2-3.)

Three special keys, CONTROL, SHIFT, and CAPS LOCK, change the codes generated by the other keys. The CONTROL key is similar to the ASCII CTRL key.

Three other keys have special functions: the RESET key, and two keys marked with apples, one outlined, or open () , and one solid, or closed (). Pressing the RESET key with the CONTROL key depressed resets the Apple IIe, as described in Chapter 4. The Apple keys are connected to the one-bit game inputs, described later in this chapter.

See Chapter 7 for a complete description of the electrical interface to the keyboard.

The electrical interface between the Apple IIe and the keyboard is a ribbon cable with a 26-pin connector. This cable carries the keyboard signals to the encoding circuitry on the main board.

Reading the Keyboard



The keyboard encoder and ROM generate all 128 ASCII codes, so all of the special character codes in the ASCII character set are available from the keyboard. Machine-language programs obtain character codes from the keyboard by reading a byte from the keyboard-data location shown in Table 2-2.





Table 2-2. Keyboard Memory Locations

Location			Description
Hex	Decimal		
\$C000	49152	-16384	Keyboard data and strobe
\$C010	49168	-16368	Any-key-down flag and clear-strobe switch

Hexadecimal refers to the base-16 number system, which uses the digits 0 through 9 and the six letters A through F to represent values from 0 to 15.

Your programs can get the code for the last key pressed by reading the keyboard-data location. Table 2-2 gives this location in three different forms: the **hexadecimal** value used in assembly language, indicated by a preceding dollar sign (\$); the decimal value used in Applesoft BASIC, and the complementary decimal value used in Apple Integer BASIC. (Integer BASIC requires that values greater than 32767 be written as the number obtained by subtracting 65536 from the value. These are the decimal numbers shown as negative in tables in this manual; refer to the *Apple II BASIC Programming Manual*.) The low-order seven bits of the byte at the keyboard location contain the character code; the high-order bit of this byte is the strobe bit, described later in this section.

Your program can find out whether any key is down, except the **RESET**, **CONTROL**, **SHIFT**, **CAPS LOCK**, , and  keys by reading from location 49168 (hexadecimal \$C010 or complementary decimal -16368). The high-order bit (bit 7) of the byte you read at this location is called any-key-down; it is 1 if a key is down, and 0 if no key is down. The value of this bit is 128; if a BASIC program gets this information with a PEEK, the value is 128 or greater if any key is down, and less than 128 if no key is down.

The  and  keys are connected to switches 0 and 1 of the game I/O connector inputs. If  is pressed, switch 0 is “pressed,” and if  is pressed, switch 1 is “pressed.”

The strobe bit is the high-order bit of the keyboard-data byte. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at the clear-strobe location. This location is a combination flag and switch; the flag tells whether any key is down, and the switch clears the strobe bit. The switch function of this memory location is called a soft switch because it is controlled by software. In this case, it doesn’t matter whether the program reads or writes, and it doesn’t matter what data the program writes: the only action that occurs is the resetting of the keyboard strobe. Similar soft switches, described later, are used for controlling other functions in the Apple IIe.

Important!

Any time you read the any-key-down flag, you also clear the keyboard strobe. If your program needs to read both the flag and the strobe, it must read the strobe bit first.

After the keyboard strobe has been cleared, it remains low until another key is pressed. Even after you have cleared the strobe, you can still read the character code at the keyboard location. The data byte has a different value, because the high-order bit is no longer set, but the ASCII code in the seven low-order bits is the same until another key is pressed. Table 2-3 shows the ASCII codes for most of the keys on the keyboard of the Apple IIe.

There are several special-function keys that do not generate ASCII codes. For example, you cannot read the **CONTROL**, **SHIFT**, and **CAPS LOCK** keys directly, but pressing one of these keys alters the character codes produced by the other keys.

Another key that doesn’t generate a code is **RESET**, located at the upper-right corner of the keyboard; it is connected directly to the Apple IIe’s circuits. Pressing **RESET** with **CONTROL** depressed normally causes the system to stop whatever program it’s running and restart itself. This restarting process is called the reset routine.

The reset routine is described in Chapter 4.



Two more special keys are the Apple keys,  and , located on either side of the **SPACE** bar. These keys are connected to the one-bit game inputs, which are described later in this chapter in the section “Switch Inputs.” Pressing them in combination with the **CONTROL** and **RESET** keys causes the built-in firmware to perform special reset and self-test cycles, described with the reset routine in Chapter 4.

Table 2-3. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
←	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
↓	0A	LF	0A	LF	0A	LF	0A	LF
↑	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
→	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29)	29)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (39	9	39	9	28	(28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] } , ~	5D 60] ,	1D 60	GS ,	7D 7E	} ~	1D 7E	GS ~

Table 2-3—Continued. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

The Video Display Generator

The primary output device of the Apple IIe is the video display. You can use any ordinary video monitor, either color or black-and-white, to display video information from the Apple IIe. An ordinary monitor is one that accepts composite video compatible with the standard set by the NTSC (National Television Standards Committee). If you use Apple IIe color graphics with a black-and-white monitor, the display will appear as black and white (or green or amber or...) and various patterns of these two shades mixed together.

If you are using only 40-column text and graphics modes, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your Apple IIe; if it does not, you'll need to attach a radio frequency (RF) video modulator between the Apple IIe and the television set.

Important!

With the 80-column text card installed, the Apple IIe can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

The specifications for the video display are summarized in Table 2-4.

Original IIe

Note that MouseText characters are not included in the original version of the Apple IIe.

The video signal produced by the Apple IIe is NTSC-compatible composite color video. It is available at three places: the RCA-type phono jack on the back of the Apple IIe, the single Molex-type pin on the main circuit board near the back on the right side, and one of the group of four Molex-type pins in the same area on the main board. Use the RCA-type phono jack to connect a video monitor or an external video modulator; use the Molex pins to connect the type of video modulator that fits inside the Apple IIe case.

For a full description of the video signal and the connections to the Molex-type pins, refer to the section "Video Output Signals" in Chapter 7.

Table 2-4. Video Display Specifications

Display modes:	40-column text; map: Figure 2-5 80-column text; map: Figure 2-6 Low-resolution color graphics; map: Figure 2-7 High-resolution color graphics; map: Figure 2-8 Double-high-res. color graphics; map: Figure 2-9
Text capacity:	24 lines by 80 columns (character positions)
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, inverse, flashing, MouseText (Table 2-5)
Low-resolution graphics:	16 colors (Table 2-6) 40 horizontal by 48 vertical; map: Figure 2-7
High-resolution graphics:	6 colors (Table 2-7) 140 horizontal by 192 vertical (restricted) Black-and-white: 280 horizontal by 192 vertical; map: Figure 2-8
Double-high-resolution graphics:	16 colors (Table 2-8) 140 horizontal by 192 vertical (no restrictions) Black-and-white: 560 horizontal by 192 vertical; map: Figure 2-9

The Apple IIe can produce seven different kinds of video display:

- ☐ text, 24 lines of 40 characters
- ☐ text, 24 lines of 80 characters (with optional text card)
- ☐ low-resolution graphics, 40 by 48, in 16 colors
- ☐ high-resolution graphics, 140 by 192, in 6 colors
- ☐ high-resolution graphics, 280 by 192, in black and white
- ☐ double high-resolution graphics, 140 by 192, in 16 colors (with optional 64K text card)
- ☐ double high-resolution graphics, 560 by 192, in black and white (with optional 64K text card)

The two text modes can display all 96 ASCII characters: the uppercase and lowercase letters, numbers, and symbols. The enhanced Apple IIe can also display MouseText characters.

Any of the graphics displays can have 4 lines of text at the bottom of the screen. The text may be either 40-column or 80-column, except that double-high-resolution graphics may only have 80-column text at the bottom of the screen. Graphics displays with text at the bottom are called mixed-mode displays.

The low-resolution graphics display is an array of colored blocks, 40 wide by 48 high, in any of 16 colors. In mixed mode, the 4 lines of text replace the bottom 8 rows of blocks, leaving 40 rows of 40 blocks each.

The high-resolution graphics display is an array of dots, 280 wide by 192 high. There are 6 colors available in high-resolution displays, but a given dot can use only 4 of the 6 colors. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 280 dots each.

The double-high-resolution graphics display uses main and auxiliary memory to display an array of dots, 560 wide by 192 high. All the dots are visible in black and white. If color is used, the display is 140 dots wide by 192 high with 16 colors available. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 560 (or 140) dots each. In mixed mode, the text lines can be 80 columns wide only.

Text Modes

The text characters displayed include the uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide, leaving two blank columns of dots between characters in a row, except for MouseText characters, some of which are seven dot wide. Except for lowercase letters with descenders and some MouseText characters, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other single color) dots on a black background. Characters can also be displayed as black dots on a white background; this is called inverse format.

Text Character Sets

The Apple IIe can display either of two text character sets: the primary set or an alternate set. The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- ☐ normal, with white dots on a black screen
- ☐ inverse, with black dots on a white screen
- ☐ flashing, alternating between normal and inverse.

With the primary character set, the Apple IIe can display uppercase characters in all three formats: normal, inverse, and flashing. Lowercase letters can only be displayed in normal format. The primary character set is compatible with most software written for the Apple II and Apple II Plus models, which can display text in flashing format but don't have lowercase characters.

The alternate character set displays characters in either normal or inverse format. In normal format, you can get

- ☐ uppercase letters
- ☐ lowercase letters
- ☐ numbers
- ☐ special characters.

In inverse format, you can get

- ☐ MouseText characters (on the enhanced Apple IIe)
- ☐ uppercase letters
- ☐ lowercase letters
- ☐ numbers
- ☐ special characters.

The MouseText characters that replace the alternate uppercase inverse characters in the range of \$40-\$5F in the original Apple IIe are inverse characters, but they don't look like it because of the way that they have been constructed.

You select the character set by means of the alternate-text soft switch, ALTCHAR, described later in the section "Display Mode Switching." Table 2-5 shows the character codes in hexadecimal for the Apple IIe primary and alternate character sets in normal, inverse, and flashing formats.

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select inverse or flashing format and uppercase or lowercase characters. In the primary character set, bit 7 selects inverse or normal format and bit 6 controls character flashing. In the alternate character set, bit 6 selects between uppercase and lowercase, according to the ASCII character codes, and flashing format is not available.

Table 2-5. Display Character Sets

Note: To identify particular characters and values, refer to Table 2-3.

Hex Values	Primary Character Set		Alternate Character Set	
	Character Type	Format	Character Type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText	
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special characters	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

Original IIE

| In the alternate character set of the original Apple IIE, characters in the range \$40-\$5F are uppercase inverse.

40-Column Versus 80-Column Text

The Apple IIE has two modes of text display: 40-column and 80-column. (The 80-column display mode described in this manual is the one you get with the Apple IIE 80-Column Text Card or other auxiliary-memory card installed in the auxiliary slot.) The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare Figure 2-2 and Figure 2-3. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

Figure 2-2. 40-Column Text Display

```
1LIST 0,100

10  REM  APPLESOFT CHARACTER DEMO

20  TEXT : HOME
30  PRINT : PRINT "Applesoft char
    acter Demo"
40  PRINT : PRINT "Which character
    set--"
50  PRINT : INPUT "Primary (P) or
    Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN POKE 49166,
    0
80  IF A$ = "A" THEN POKE 49167,
    0
90  PRINT : PRINT "...printing th
    e same line, first"
100 PRINT " in NORMAL, then INVE
    RSE ,then FLASH:": PRINT
1
```

Figure 2-3. 80-Column Text Display

```
1LIST

10  REM APPLESOFT CHARACTER DEMO
20  TEXT : HOME
30  PRINT : PRINT "Applesoft Character Demo"
40  PRINT : PRINT "Which character set--"
50  PRINT : INPUT "Primary (P) or Alternate (A) ?";A$
60  IF LEN (A$) < 1 THEN 50
65  LET A$ = LEFT$ (A$,1)
70  IF A$ = "P" THEN POKE 49166,0
80  IF A$ = "A" THEN POKE 49167,0
90  PRINT : PRINT "...printing the same line, first"
100 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
150 NORMAL : GOSUB 1000
160 INVERSE : GOSUB 1000
170 FLASH : GOSUB 1000
180 NORMAL : PRINT : PRINT : PRINT "Press any key to repeat."
190 GET A$
200 GOTO 10
1000 PRINT : PRINT "SAMPLE TEXT: Now is the time--12:00"
1100 RETURN
1■
```

Graphics Modes

The Apple IIe can produce video graphics in three different modes. All the graphics modes treat the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes in these arrays; this section describes the way the resulting graphics data are stored in the Apple IIe's memory.

Low-Resolution Graphics

In the low-resolution graphics mode, the Apple IIe displays an array of 48 rows by 40 columns of colored blocks. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and three shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

Data for the low-resolution graphics display is stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 2-6. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value \$D8 produces a brown block atop a yellow block on the screen.

Table 2-6. Low-Resolution Graphics Colors

Note: Colors may vary, depending upon the controls on the monitor or TV set.

Nibble Value			Nibble Value		
Dec	Hex	Color	Dec	Hex	Color
0	\$00	Black	8	\$08	Brown
1	\$01	Magenta	9	\$09	Orange
2	\$02	Dark Blue	10	\$0A	Gray 2
3	\$03	Purple	11	\$0B	Pink
4	\$04	Dark Green	12	\$0C	Light Green
5	\$05	Gray 1	13	\$0D	Yellow
6	\$06	Medium Blue	14	\$0E	Aquamarine
7	\$07	Light Blue	15	\$0F	White

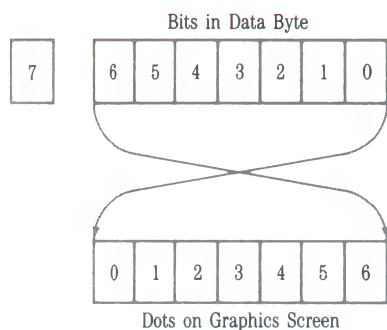
As explained later in the section “Display Pages,” the text display and the low-resolution graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display it as graphics, or vice-versa. All you have to do is change the mode switch, described later in this chapter in the section “Display Mode Switching,” without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex low-resolution graphics displays quickly.

High-Resolution Graphics

In the high-resolution graphics mode, the Apple IIe displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described later in this section. Adjacent dots of the same color merge to form a larger colored area.

Data for the high-resolution graphics displays are stored in either of two 8192-byte areas in memory. These areas are called high-resolution Page 1 and Page 2; think of them as buffers where you can put data to be displayed. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Apple IIe's memory.

Figure 2-4. High-Resolution Display Bits



The Apple IIe high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Apple IIe's memory. The seven low-order bits of each display byte control a row of seven adjacent dots on the screen, and forty adjacent bytes in memory control a row of 280 (7 times 40) dots. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the second-least significant bit, and so on, as shown in Figure 2-4. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described later.

On a black-and-white monitor, there is a simple correspondence between bits in memory and dots on the screen. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots blur together; alternating black and white dots merge to a continuous grey.

On an NTSC color monitor or a color television set, a dot whose controlling bit is off (0) is black. If the bit is on, the dot will be white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte.

Call the left-most column of dots column zero, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control dots in even-numbered columns (0, 2, 4, and so forth) are on, the dots are purple; if the bits that control odd-numbered columns are on, the dots are green—but only if the dots on both sides of a given dot are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange—again, only if the dots on both sides are black. Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In other words, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- ☐ Dots in even columns can be black, purple, or blue.
- ☐ Dots in odd columns can be black, green, or orange.
- ☐ If adjacent dots in a row are both on, they are both white.
- ☐ The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

For more details about the way the Apple IIe produces color on a TV set, see the section "Video Display Modes" in Chapter 7.

These rules are summarized in Table 2-7. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 2-7. High-Resolution Graphics Colors

Note: Colors may vary depending upon the controls on the monitor or television set.

Bits 0-6	Bit 7 Off	Bit 7 On
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

For information about the way NTSC color television works, see the magazine articles listed in the bibliography.

The peculiar behavior of the high-resolution colors reflects the way NTSC color television works. The dots that make up the Apple IIe video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating black and white dots at this spacing cause a color monitor or TV set to produce color, but two or more white dots together do not.

Double-High-Resolution Graphics

Double-high-resolution graphics is a bit-mapping of the low-order seven bits of the bytes in the main-memory and auxiliary-memory pages at \$2000-\$3FFF. The bytes in the main-memory and auxiliary-memory pages are interleaved in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. Horizontal resolution is 560 dots when displayed on a monochrome monitor.

Unlike high-resolution color, double-high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a 4-dot-wide window moving across the screen: at any given time, the color displayed will correspond to the 4-bit value from Table 2-8 that corresponds to the window's position (Figure 2-9). Effective horizontal resolution with color is 140 (560 divided by four) dots per line.

To use Table 2-8, divide the display column number by 4, and use the remainder to find the correct column in the table: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on); *mb1* is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 5, 9 and so on), and similarly for *ab3* and *mb4*.

Table 2-8. Double-High-Resolution Graphics Colors

Color	ab0	mb1	ab2	mb3	Repeated Bit Pattern
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark Green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$55	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark Blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium Blue	\$33	\$66	\$4C	\$19	1100
Light Blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

Video Display Pages

The Apple IIe generates its video displays using data stored in specific areas in memory. These areas, called display pages, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object at a certain location on the display. In text mode, the object is a single character; in low-resolution graphics, the object is two stacked colored blocks; and in high-resolution and double-high-resolution modes, it is a line of seven adjacent dots.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called text Page 1 and text Page 2, and they are located at 1024-2047 (hexadecimal \$0400-\$07FF) and 2048-3071 (\$0800-\$0BFF) in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch displays instantly. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four rows of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory located on the 80-column text card. This additional memory is *not* the same as text Page 2—in fact, it occupies the same address space as text Page 1, and there is a special soft switch that enables you to store data into it. (See the next section “Display Mode Switching.”) The built-in firmware I/O routines described in Chapter 3 take care of this extra addressing automatically; that is one reason to use those routines for all your normal text output.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage, as shown in Table 2-9.

The double-high-resolution graphics mode uses high-resolution Page 1 in both main and auxiliary memory. Each byte in those pages of memory controls a display area seven dots wide by one dot high. This gives you 560 dots per line in black and white, and 140 dots per line in color. A double-high-resolution display requires twice the total memory as high-resolution graphics, and 16 times as much as a low-resolution display.

Table 2-9. Video Display Page Locations

Display Mode	Display Page	Lowest Address		Highest Address	
		Hex	Dec	Hex	Dec
40-column text, low-resolution graphics	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
80-column text	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
High-resolution graphics	1	\$2000	8192	\$3FFF	16383
	2	\$4000	16384	\$5FFF	24575
Double-high- resolution graphics	1 †	\$2000	8192	\$3FFF	16383
	2 †	\$4000	16384	\$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to the next section “Display Mode Switching.”

† See the section “Double-High-Resolution Graphics,” earlier in this chapter.

Display Mode Switching

You select the display mode that is appropriate for your application by reading or writing to a reserved memory location called a soft switch. In the Apple IIe, most soft switches have three memory locations reserved for them: one for turning the switch on, one for turning it off, and one for reading the current state of the switch.

Table 2-10 shows the reserved locations for the soft switches that control the display modes. For example, to switch from mixed-mode to full-screen graphics in an assembly-language program, you could use the instruction

```
STA      $C052
```

To do this in a BASIC program, you could use the instruction

```
POKE    49234,0
```

Some of the soft switches in Table 2-10 must be read, some must be written to, and for some you can use either action. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

Table 2-10. Display Soft Switches

Note: W means write anything to the location, R means read the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
ALTCHAR	W	\$C00E	Off: display text using primary character set
ALTCHAR	W	\$C00F	On: display text using alternate character set
RDALTCHAR	R7	\$C01E	Read ALTCHAR switch (1 = on)
80COL	W	\$C00C	Off: display 40 columns
80COL	W	\$C00D	On: display 80 columns
RD80COL	R7	\$C01F	Read 80COL switch (1 = on)
80STORE	W	\$C000	Off: cause PAGE2 on to select auxiliary RAM
80STORE	W	\$C001	On: allow PAGE2 to switch main RAM areas
RD80STORE	R7	\$C018	Read 80STORE switch (1 = on)
PAGE2	R/W	\$C054	Off: select Page 1
PAGE2	R/W	\$C055	On: select Page 2 or, if 80STORE on, Page 1 in auxiliary memory
RDPAGE2	R7	\$C01C	Read PAGE2 switch (1 = on)
TEXT	R/W	\$C050	Off: display graphics or, if MIXED on, mixed
TEXT	R/W	\$C051	On: display text
RDTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C052	Off: display only text or only graphics
MIXED	R/W	\$C053	On: if TEXT off, display text and graphics
RDMIXED	R7	\$C01B	Read MIXED switch (1 = on)
HIRES	R/W	\$C056	Off: if TEXT off, display low-resolution graphics
HIRES	R/W	\$C057	On: if TEXT off, display high-resolution or, if DHIRES on, double-high-resolution graphics
RDHIRES	R7	\$C01D	Read HIRES switch (1 = on)
IOUDIS	W	\$C07E	On: disable IOU access for addresses \$C058 to \$C05F; enable access to DHIRES switch *
IOUDIS	W	\$C07F	Off: enable IOU access for addresses \$C058 to \$C05F; disable access to DHIRES switch *
RДИОUDIS	R7	\$C07E	Read IOUDIS switch (1 = off) †
DHIRES	R/W	\$C05E	On: if IOUDIS on, turn on double-high-res.
DHIRES	R/W	\$C05F	Off: if IOUDIS on, turn off double-high-res.
RDDHIRES	R7	\$C07F	Read DHIRES switch (1 = on) †

* The firmware normally leaves IOUDIS on. See also †.

† Reading or writing any address in the range \$C070-\$C07F also triggers the paddle timer and resets VBLINT (Chapter 7).

By the Way: You may not need to deal with these functions by reading and writing directly to the memory locations in Table 2-10. Many of the functions shown here are selected automatically if you use the display routines in the various high-level languages on the Apple IIe.

Any time you read a soft switch, you get a byte of data. However, the only information the byte contains is the state of the switch, and this occupies only one bit—bit 7, the high-order bit. The other bits in the byte are unpredictable. If you are programming in machine language, the switch setting is the sign bit; as soon as you read the byte, you can do a Branch Plus if the switch is off, or Branch Minus if the switch is on.

If you read a soft switch from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if the switch is on, the value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

Addressing Display Pages Directly

Before you decide to use the display pages directly, consider the alternatives. Most high-level languages enable you to write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you may be able to use the display features of the built-in I/O firmware. You should store directly into display memory only if the existing programs can't meet your requirements.

For a full description of the way the Apple IIe handles its display memory, refer to the section "Display Memory Addressing" in Chapter 7.

The display memory maps are shown in Figures 2-5, 2-6, 2-7, 2-8, and 2-9. All of the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hexadecimal). By folding the display data into memory this way, the Apple IIe, like the Apple II, stores all 960 characters of displayed text within 1K bytes of memory.

The high-resolution graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the high-resolution display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or 8192 bytes. In other words, each block of 1024 bytes in the high-resolution display page contains one row of dots out of every group of eight rows. The individual rows are stored in sets of three 40-byte rows, the same way as the text display.

All of the display modes except 80-column mode and double-high-resolution graphics mode can use either of two display pages. The display maps show addresses for each mode's Page 1 only. To obtain addresses for text or low-resolution graphics Page 2, add 1024 (\$400); to obtain addresses for high-resolution Page 2, add 8192 (\$2000).

The 80-column display and double-high-resolution graphics mode work a little differently. Half of the data is stored in the normal text Page-1 memory, and the other half is stored in memory on the 80-column text card using the same addresses. The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the 80-column text card memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the 80-column text card memory stores the characters in the even columns.

For more details about the way the displays are generated, see Chapter 7.

To store display data on the 80-column text card, first turn on the 80STORE soft switch by writing to location 49153 (hexadecimal \$C001 or complementary -16383). With 80STORE on, the page-select switch, PAGE2, selects between the portion of the 80-column display stored in Page 1 of main memory and the portion stored in the 80-column text card memory. To select the 80-column text card, turn the PAGE2 soft switch on by reading or writing at location 49237.

Figure 2-5. Map of 40-Column Text Display

			\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F	\$10	\$11	\$12	\$13	\$14	\$15	\$16	\$17	\$18	\$19	\$1A	\$1B	\$1C	\$1D	\$1E	\$1F	\$20	\$21	\$22	\$23	\$24	\$25	\$26	\$27	
Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39			
0	\$400	1024																																									
1	\$480	1152																																									
2	\$500	1280																																									
3	\$580	1408																																									
4	\$600	1536																																									
5	\$680	1664																																									
6	\$700	1792																																									
7	\$780	1920																																									
8	\$428	1064																																									
9	\$4A8	1192																																									
10	\$528	1320																																									
11	\$5A8	1448																																									
12	\$628	1576																																									
13	\$6A8	1704																																									
14	\$728	1832																																									
15	\$7A8	1960																																									
16	\$450	1104																																									
17	\$4D0	1232																																									
18	\$550	1360																																									
19	\$5D0	1488																																									
20	\$650	1616																																									
21	\$6D0	1744																																									
22	\$750	1872																																									
23	\$7D0	2000																																									

Figure 2-6. Map of 80-Column Text Display

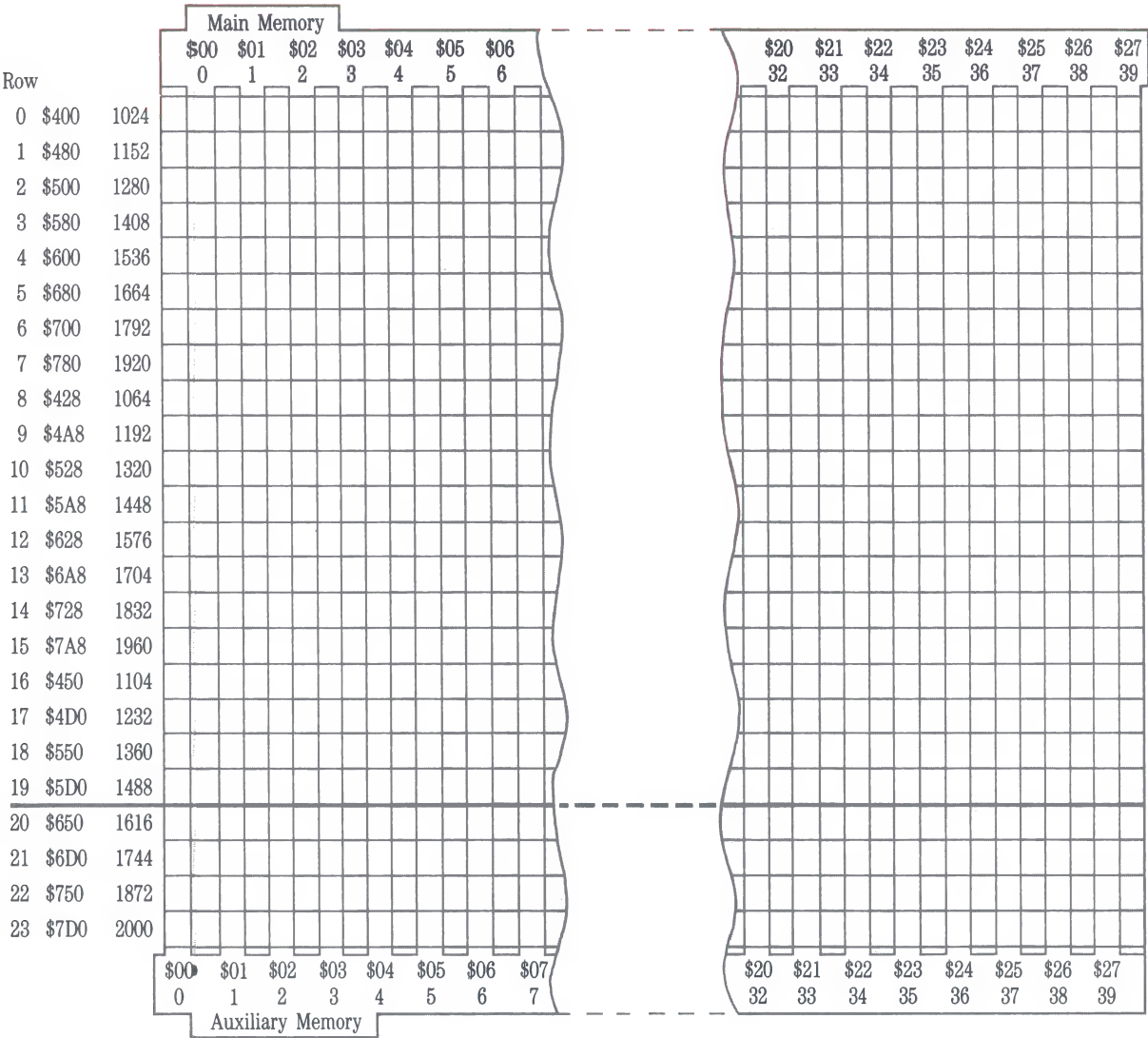


Figure 2-7. Map of Low-Resolution Graphics Display

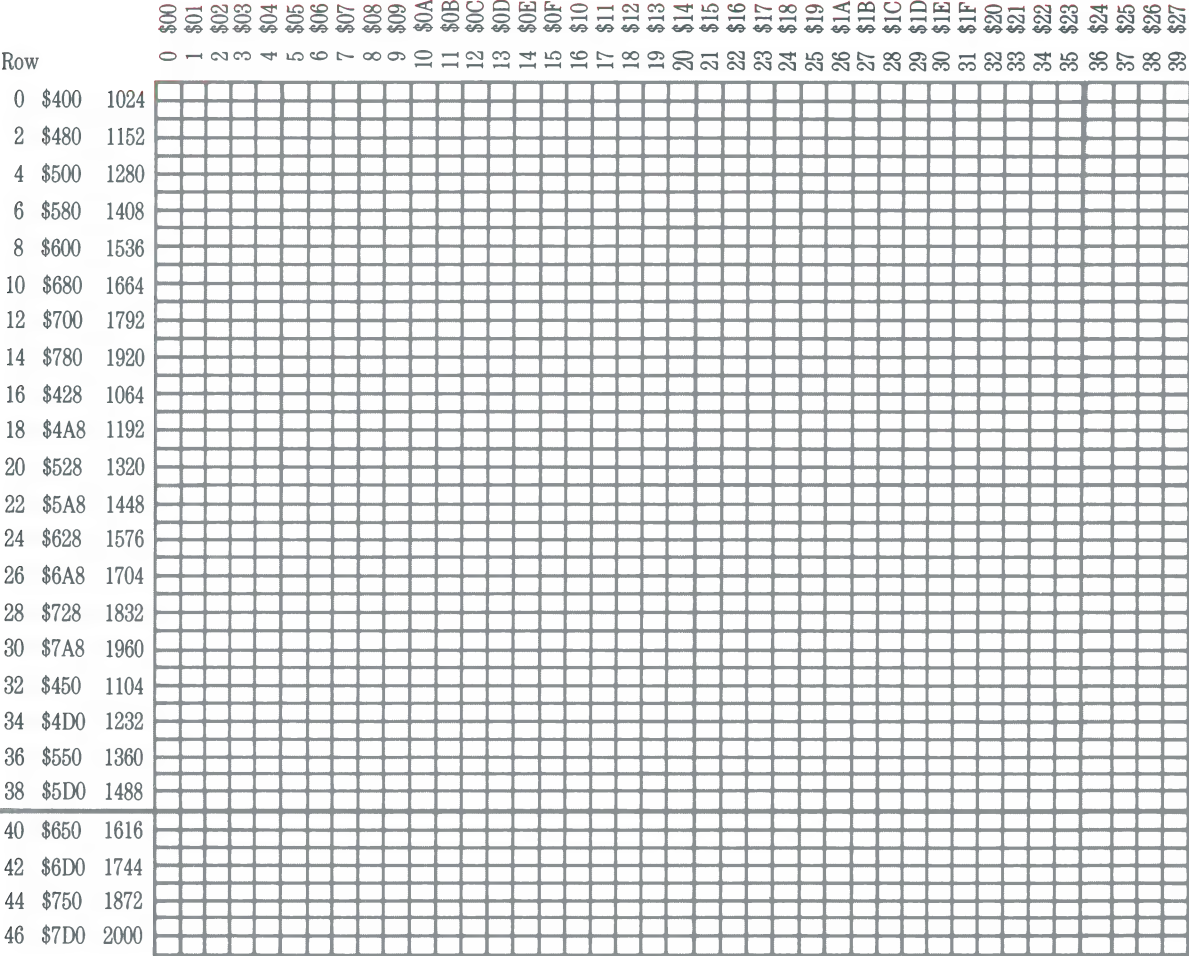


Figure 2-8. Map of High-Resolution Graphics Display

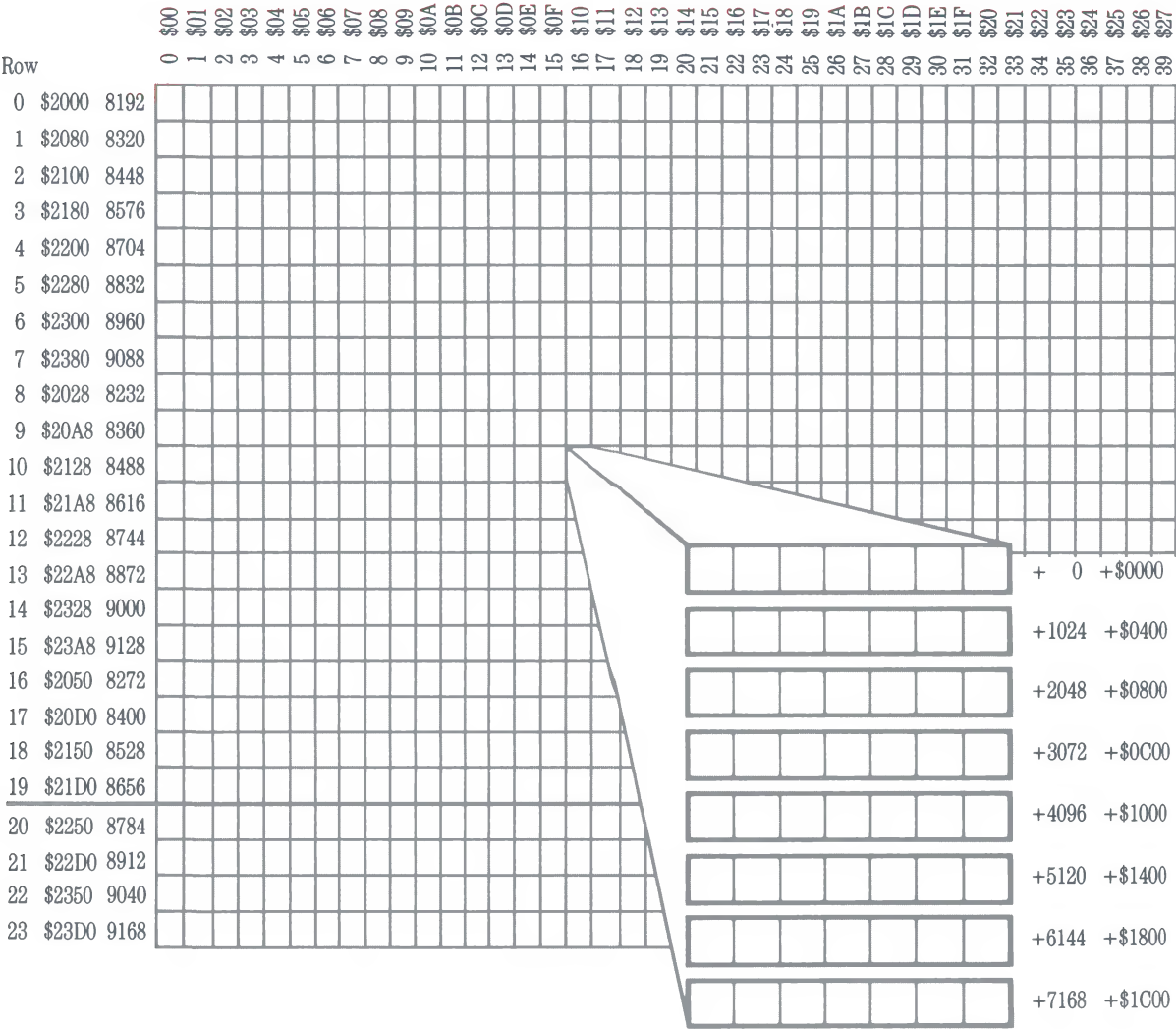
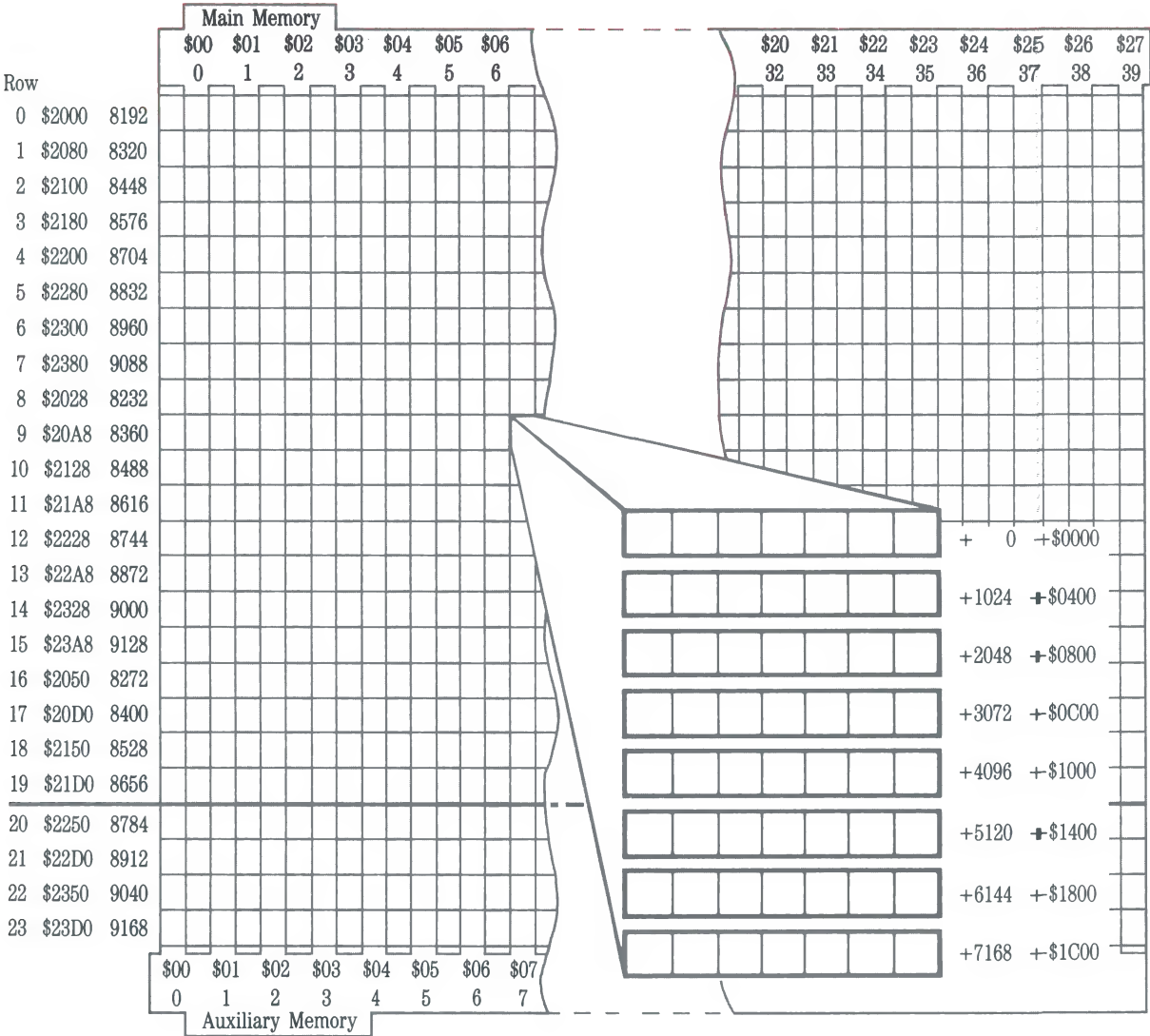


Figure 2-9. Map of Double-High-Resolution Graphics Display



Secondary Inputs and Outputs

In addition to the primary I/O devices—the keyboard and display—there are several secondary input and output devices in the Apple IIe. These devices are

- ❑ the speaker (output)
- ❑ cassette input and output
- ❑ annunciator outputs
- ❑ strobe output
- ❑ switch inputs
- ❑ analog (hand control) inputs.

These devices are similar in operation to the soft switches described in the previous section: you control them by reading or writing to dedicated memory locations. Action takes place any time your program reads or writes to one of these locations; information written is ignored.

Important!

Some of these devices toggle—change state—each time they are accessed. If you write using an indexed store operation, the Apple IIe's microprocessor activates the address bus twice during successive clock cycles, causing a device that toggles each time it is addressed to end up back in its original state. For this reason, you should read, rather than write, to such devices.

The Speaker

Electrical specifications of the speaker circuit appear in Chapter 7.

The Apple IIe has a small speaker mounted toward the front of the bottom plate. The speaker is connected to a soft switch that toggles; it has two states, off and on, and it changes from one to the other each time it is accessed. (At low frequencies, less than 400 Hz or so, the speaker clicks only on every other access.)

If you switch the speaker once, it emits a click; to make longer sounds, you access the speaker repeatedly. You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.

BELL1 is described in Appendix B.

The soft switch for the speaker uses memory location 49200 (hexadecimal \$C030). From Integer BASIC, use the complementary address -16336. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program. There is also a routine in the built-in firmware to make a beep through the speaker. This routine is called BELL1.

Cassette Input and Output

There are two miniature phone jacks on the back panel of the Apple IIe. You can use a pair of standard cables with miniature phone plugs to connect an ordinary cassette tape recorder to the Apple IIe and save programs and data on audio cassettes.

Detailed electrical specifications for the cassette input and output are given in Chapter 7.

The phone jack marked with a picture of an arrow pointing towards a cassette is the output jack. It is connected to a toggled soft switch, like the speaker switch described above. The signal at the phone jack switches from zero to 25 millivolts or from 25 millivolts to zero each time you access the soft switch.

If you connect a cable from this jack to the microphone input of a cassette tape recorder and switch the recorder to record mode, the signal changes you produce by accessing this soft switch will be recorded on the tape. The cassette output switch uses memory location 49184 (hexadecimal \$C020; complementary value -16352). Like the speaker, this output will toggle twice if you write to it, so you should only use read operations to control the cassette output.

WRITE is described in Appendix B.

The standard method for writing computer data on audio tapes uses tones with two different pitches to represent the binary states zero and one. To store data, you convert the data into a stream of bits and convert the bits into the appropriate tones. To save you the trouble of actually programming the tones, and to ensure consistency among all Apple II cassette tapes, there is a built-in routine called WRITE for producing cassette data output.

The phone jack marked with a picture of an arrow coming from a cassette is the input jack. It accepts a cable from the cassette recorder's earphone jack. The signal from the cassette is 1 volt (peak-to-peak) audio. Each time the instantaneous value of this audio signal changes from positive to negative, or vice-versa, the state of the cassette input circuit changes from zero to one or vice-versa. You can read the state of this circuit at memory location 49248 (hexadecimal \$C060, or complementary decimal -16288).

When you read this location, you get a byte, but only the high-order bit (bit 7) is valid. If you are programming in machine language, this is the sign bit, so you can perform a Branch Plus or Branch Minus immediately after reading this byte. BASIC is too slow to keep up with the audio tones used for data recording on tape, but you don't need to write the program: there is a built-in routine called READ for reading data from a cassette.

READ is described in Appendix B.

The Hand Control Connector Signals

Several inputs and outputs are available on a 9-pin D-type miniature connector on the back of the Apple IIe: three one-bit inputs, or switches, and four analog inputs. These signals are also available on the 16-pin IC connector on the main circuit board, along with four one-bit outputs and a data strobe. You can access all of these signals from your programs.

Complete electrical specifications of these inputs and outputs are given in Chapter 7.

Ordinarily, you connect a pair of hand controls to the 9-pin connector. The rotary controls use two analog inputs, and the push-buttons use two one-bit inputs. However, you can also use these inputs and outputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick. Table 7-19 shows the connector pin numbers.

For electrical specifications of the annunciator outputs, refer to Chapter 7.

Annunciator Outputs

The four one-bit outputs are called annunciators. Each annunciator can be used to turn a lamp, a relay, or some similar electronic device on and off.

Each annunciator is controlled by a soft switch, and each switch uses a pair of memory locations. These memory locations are shown in Table 2-11. Any reference to the first location of a pair turns the corresponding annunciator off; a reference to the second location turns the annunciator on. There is no way to read the state of an annunciator.

Table 2-11. Annunciator Memory Locations

No.	Annunciator		Address		
	Pin*	State	Decimal		Hex
0	15	off	49240	-16296	\$C058
		on	49241	-16295	\$C059
1	14	off	49242	-16294	\$C05A
		on	49243	-16293	\$C05B
2	13	off	49244	-16292	\$C05C
		on	49245	-16291	\$C05D
3	12	off	49246	-16290	\$C05E
		on	49247	-16289	\$C05F



* Pin numbers given are for the 16-pin IC connector on the circuit board.

Strobe Output

The strobe output is normally at +5 volts, but it drops to zero for about half a microsecond any time its dedicated memory location is accessed. You can use this signal to control functions such as data latching in external devices. If you use this signal, remember that memory is addressed twice by a write; if you need only a single pulse, use a read operation to activate the strobe. The memory location for the strobe signal is 49216 (hexadecimal \$C040 or complementary -16320).

Switch Inputs

The three one-bit inputs can be connected to the output of another electronic device or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you read the switch with a PEEK and compare the value with 128. If the value is 128 or greater, the switch is on.

The memory locations for these switches are 49249 through 49251 (hexadecimal \$C061 through \$C063, or complementary -16287 through -16285), as shown in Table 2-12. Switch 0 and switch 1 are permanently connected to the  and  keys on the keyboard; these are the ones normally connected to the buttons on the hand controls. Some software for the older models of the Apple II uses the third switch, switch 2, as a way of detecting the shift key. This technique requires a hardware modification known as the single-wire shift-key mod.

You should be sure that you really need the shift-key mod before you go ahead and do it. It probably is not worth it unless you have a program that requires the shift-key mod that you cannot either replace or modify to work without it.

▲ Warning

If you make the shift-key modification and connect a joystick or other hand control that uses switch 2, you must be careful never to close the switch and press **SHIFT** at the same time: doing so produces a short circuit that causes the power supply to turn off. When this happens, any programs or data in the computer's internal memory are lost.

Shift-Key Mod: To perform this modification on your Apple IIe, all you have to do is solder across the broken diamond labelled X6 on the main circuit board. Remember to turn off the power before changing anything inside the Apple IIe. Also remember that changes such as this are at your own risk and may void your warranty.

Analog Inputs

Refer to the section “Game I/O Signals” in Chapter 7 for details.

The four analog inputs are designed for use with 150K ohm variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit. The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

Before a program can read the analog inputs, it must first reset the timing circuits. Accessing memory location 49264 (hexadecimal \$C070 or complementary -16272) does this. As soon as you reset the timing circuits, the high bits of the bytes at locations 49252 through 49255 (hexadecimal \$C064 through \$C067 or complementary -16284 through -16281) are set to 1. If you PEEK at them from BASIC, the values will be 128 or greater. Within about 3 milliseconds, these bits will change back to 0—byte values less than 128—and remain there until you reset the timing circuits again. The exact time each of the four bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

PREAD is described in Appendix B.

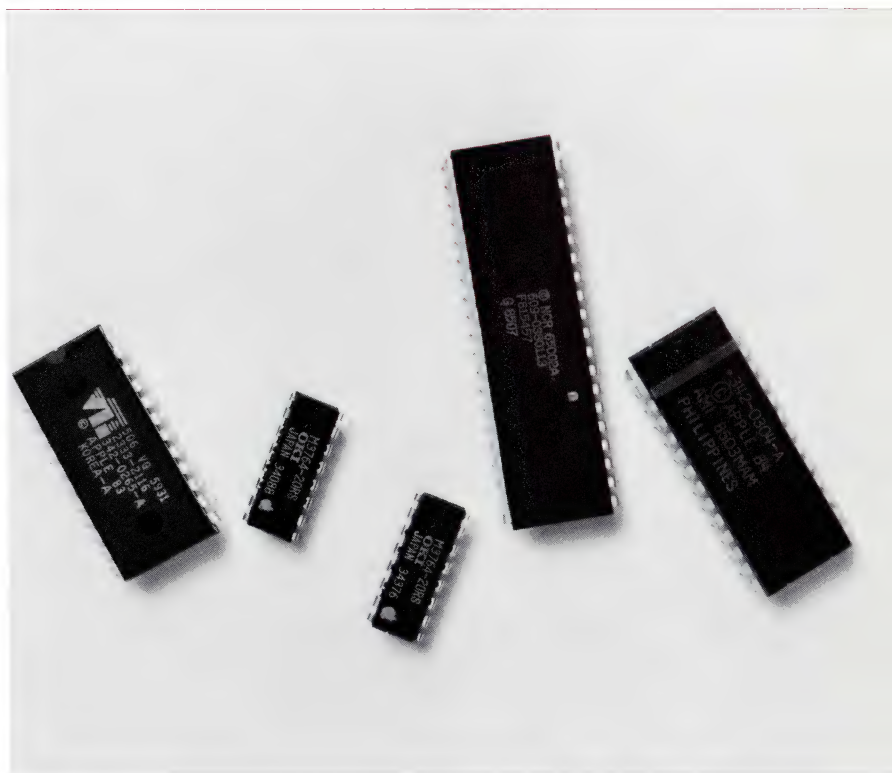
To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to 0, or you can use the built-in routine called PREAD. High-level languages, such as BASIC, also include convenient means of reading the analog inputs: refer to your language manuals.

Summary of Secondary I/O Locations

Table 2-12 shows the memory locations for all of the built-in I/O devices except the keyboard and display. As explained earlier, some soft switches should only be accessed by means of read operations; those switches are marked.

Table 2-12. Secondary I/O Memory Locations*For connector identification and pin numbers, refer to Tables 7-18 and 7-19.*

Function	Address		Hex	Access
	Decimal			
Speaker	49200	-16336	\$C030	Read only
Cassette out	49184	-16352	\$C020	Read only
Cassette in	49248	-16288	\$C060	Read only
Annunciator 0 on	49241	-16295	\$C059	
Annunciator 0 off	49240	-16296	\$C058	
Annunciator 1 on	49243	-16293	\$C05B	
Annunciator 1 off	49242	-16294	\$C05A	
Annunciator 2 on	49245	-16291	\$C05D	
Annunciator 2 off	49244	-16292	\$C05C	
Annunciator 3 on	49247	-16289	\$C05F	
Annunciator 3 off	49246	-16290	\$C05E	
Strobe output	49216	-16320	\$C040	Read only
Switch input 0 (☐)	49249	-16287	\$C061	Read only
Switch input 1 (☐)	49250	-16286	\$C062	Read only
Switch input 2	49251	-16285	\$C063	Read only
Analog input reset	49264	-16272	\$C070	
Analog input 0	49252	-16284	\$C064	Read only
Analog input 1	49253	-16283	\$C065	Read only
Analog input 2	49254	-16282	\$C066	Read only
Analog input 3	49255	-16281	\$C067	Read only



The **Monitor**, or System Monitor, is a computer program that is used to operate the computer at the machine language level.

Almost every program on the Apple IIe takes input from the keyboard and sends output to the display. The **Monitor** and the Applesoft and Integer BASICs do this by means of standard I/O subroutines that are built into the Apple IIe's firmware. Many application programs also use the standard I/O subroutines, but Pascal programs do not; Pascal has its own I/O subroutines.

This chapter describes the features of these subroutines as they are used by the Monitor and by the BASIC interpreters, and tells you how to use the standard subroutines in your assembly-language programs.

Important! High-level languages already include convenient methods for handling most of the functions described in this chapter. You should not need to use the standard I/O subroutines in your programs unless you are programming in assembly language.

Table 3-1. Monitor Firmware Routines

Location	Name	Description
\$C305	BASICIN	With 80-column dirmware active, displays solid, blinking cursor. Accepts character from keyboard.
\$C307	BASICOUT	Displays a character on the screen; used w hen the 80-column firmware is active (Chapter 3).
\$FC9C	CLREOL	Clears to end of line from current cursor position.
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position.
\$FC42	CLREOP	Clears to bottom of window.
\$F832	CLRSCR	Clears the low-resolution screen.
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen.
\$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3).
\$FDF0	COUT1	Displays a character on the screen (Chapter 3).
\$FD8E	CROUT	Generates a carriage return character.
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character.
\$FD6A	GETLN	Displays the prompt character; accepts a string of characters by means of RDKEY.
\$F819	HLINE	Draws a horizontal line of blocks.
\$FC58	HOME	Clears the window and puts cursor in upper-left corner of window.

Table 3-1—Continued. Monitor Firmware Routines

Location	Name	Description
\$FD1B	KEYIN	With 80-column firmware inactive, displays checkerboard cursor. Accepts character from keyboard.
\$F800	PLOT	Plots a single low-resolution block on the screen.
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device.
\$FDDA	PRBYTE	Prints a hexadecimal byte.
\$FF2D	PRERR	Sends ERR and Control-G to the output device.
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number.
\$F941	PRNTAX	Prints contents of A and X in hexadecimal.
\$FD0C	RDKEY	Displays blinking cursor; goes to standard input routine, normally KEYIN or BASICIN.
\$F871	SCRN	Reads color value of a low-resolution block.
\$F864	SETCOL	Sets the color for plotting in low-resolution.
\$FC24	VTABZ	Sets cursor vertical position.
\$F828	VLINE	Draws a vertical line of low-resolution blocks.

AUXMOVE and XFER are described in the section “Auxiliary-Memory Subroutines” in Chapter 4.

The standard I/O subroutines listed in Table 3-1 are fully described in this chapter. The Apple IIe firmware also contains many other subroutines that you might find useful. Those subroutines are described in Appendix B. Two of the built-in subroutines, AUXMOVE and XFER, can help you use the optional auxiliary memory.

Using the I/O Subroutines

Before you use the standard I/O subroutines, you should understand a little about the way they are used. The Apple IIe firmware operates differently when an option such as an 80-column text card is used. This section describes general situations that affect the operation of the standard I/O subroutines. Specific instances are described in the sections devoted to the individual subroutines.

Apple II Compatibility

Compared to older Apple II models, the Apple IIe has some additional keyboard and display features. To run programs that were written for the older models, you can make the Apple IIe resemble an Apple II Plus by turning those features off. The features that you can turn off and on to put the Apple IIe into and out of Apple II mode are listed in Table 3-2.

Table 3-2. Apple II Mode

	Apple IIe	Apple II Mode
Keyboard	Uppercase and lowercase	Uppercase only
Display characters	Inverse and normal only	Flashing, inverse, and normal
Display size	40-column; also 80-column with optional card	40-column only

If the Apple IIe does not have an 80-column text card installed in the auxiliary slot, it is almost in Apple II mode as soon as you turn it on or reset it. One exception is the keyboard, which is both uppercase and lowercase.

Original IIe

On an original Apple IIe, DOS 3.3 commands and statements in Integer BASIC and Applesoft must be typed in uppercase letters. To be compatible with older software, you should switch the Apple IIe keyboard to uppercase by pressing **CAPS LOCK**.

Another feature that is different on the Apple IIe as compared to the Apple II is the displayed character set. An Apple II displays only uppercase characters, but it displays them three ways: normal, inverse, and flashing. The Apple IIe can display uppercase characters all three ways, and it can display lowercase characters in the normal way. This combination is called the *primary character set*. When the Apple IIe is first turned on or reset, it displays the primary character set.

The Apple IIe has another character set, called the *alternate character set*, that displays a full set of normal and inverse characters, with the inverse uppercase characters between \$40 and \$5F replaced on enhanced Apple IIe's with MouseText characters.

Original IIe

In the original Apple IIe, uppercase inverse characters appear in place of the MouseText characters of the enhanced Apple IIe and the Apple IIc.

You can switch character sets at any time by means of the ALTCHAR soft switch.

The primary and alternate character sets are described in Chapter 2 in the section "Text Character Sets."

The ALTCHAR soft switch is described in Chapter 2.

The 80-Column Firmware

There are a few features that are normally available only with the optional 80-column display. These features are identified in Table 3-3b and Table 3-6. The firmware that supports these features is built into the Apple IIe, but it is normally active only if an 80-column text card is installed in the auxiliary slot.

When you turn on power or reset the Apple IIe, the 80-column firmware is inactive and the Apple IIe displays the primary character set, even if an 80-column text card is installed. When you activate the 80-column firmware, it switches to the alternate character set.

The built-in 80-column firmware is implemented as if it were installed in expansion slot 3. Programs written for an Apple II or Apple II Plus with an 80-column text card installed in slot 3 usually will run properly on a Apple IIe with an 80-column text card in the auxiliary slot.

See the section “Switching I/O Memory” in Chapter 6 for details.

If the Apple IIe has an 80-column text card and you want to use the 80-column display, you can activate the built-in firmware from BASIC by typing

PR#3

To activate the 80-column firmware from the Monitor, press **[3]**, then **[CONTROL]-[P]**. Notice that this is the same procedure you use to activate a card in expansion slot 3. Any card installed in the auxiliary slot takes precedence over a card installed in expansion slot 3:

Important!

Even though you activated the 80-column firmware by typing PR#3, you should never deactivate it by typing PR#0, because that just disconnects the firmware, leaving several soft switches still set for 80-column operation. Instead, type the sequence **[ESC] [CONTROL]-[Q]**. (See Table 3-6.)

SLOT3ROM is described in Chapter 6 in the section “Switching I/O Memory.”

If there is no 80-column text card or other auxiliary memory card in your Apple IIe, you can still activate the 80-column firmware and use it with a 40-column display. First, set the SLOT3ROM soft switch located at \$C00A (49162). Then type PR#3 to transfer control to the firmware.

For more information about interrupts, see Chapter 6.

When the 80-column firmware is active without a card in the auxiliary slot, it does not work quite the same as it does with a card. The functions that clear the display (CLREOL, CLEOLZ, CLREOP, and HOME) work as if the firmware were inactive: they always clear to the current color. Also, interrupts are supported only with a card installed in the auxiliary slot.

▲Warning

If you do not have an interface card in either the auxiliary slot or slot 3, don't try to activate the firmware with PR#3. Typing PR#3 with no card installed transfers control to the empty connector, with unpredictable results.

Programs activate the 80-column firmware by transferring control to address \$C300. If there is no card in the auxiliary slot, you must set the SLOTC3ROM soft switch first. To deactivate the 80-column firmware from a program, write a Control-U character via subroutine COUT.

The Old Monitor

Apple II's and Apple II Pluses used a version of the System Monitor different from the one the Apple IIe uses. It had the same standard I/O subroutines, but a few of the features were different; for example, there were no arrow keys for cursor motion. If you start the Apple IIe with a DOS or BASIC disk that loads Integer BASIC into the bank-switched area in RAM, the old Monitor (sometimes called the Autostart Monitor) is also loaded with it. When you type INT from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type FP to switch back to Applesoft, which uses the new Monitor in ROM, or type

PR#3

to activate the 80-column firmware. Part of the firmware's initialization procedure checks to see which version of the Monitor is in RAM. If it finds the old Monitor, it replaces it with a copy of the new Monitor from ROM. After the firmware has copied the new Monitor into RAM, it remains there until the next time you start up the system.

The Standard I/O Links

When you call one of the character I/O subroutines (COUT and RDKEY), the first thing that happens is an indirect jump to an address stored in programmable memory. Memory locations used for transferring control to other subroutines are sometimes called vectors; in this manual, the locations used for transferring control to the I/O subroutines are called **I/O links**. In a Apple IIe running without a disk operating system, each I/O link is normally the address of the body of the subroutine (COUT1 or KEYIN). If a disk operating system is running, one or both of these links hold the addresses of the corresponding DOS or ProDOS I/O routines instead. (DOS and ProDOS maintain their own links to the standard I/O subroutines.)

For more information about the I/O links, see the section “Changing the Standard I/O Links” in Chapter 6.

By calling the I/O subroutines that jump to the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly in conjunction with other software, such as DOS or a printer driver, that changes one or both of the I/O links.

For the purposes of this chapter, we shall assume that the I/O links contain the addresses of the standard I/O subroutines—COUT1 and KEYIN if the 80-column firmware is off, and BASICOUT and BASICIN if it is on.

Standard Output Features

The standard output routine is named COUT, pronounced C-out, which stands for *character out*. COUT normally calls COUT1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COUT1 restricts its use of the display to an active area called the text window, described below.

COUT Output Subroutine

Your program makes a subroutine call to COUT at memory location \$FDED with a character in the accumulator. COUT then passes control via the output link CSW to the current output subroutine, normally COUT1 (or BASICOUT), which takes the character in the accumulator and writes it out. If the accumulator contains an uppercase or lowercase letter, a number, or a special character, COUT1 displays it; if the accumulator contains a control character, COUT1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COUT1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right-hand edge of the window, COUT1 moves it to the left-most position on the next line down. If this would move the cursor position past the end of the last line in the window, COUT1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations 36 and 37 (hexadecimal \$24 and \$25). These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COUT1 does not display a cursor, but the input routines described below do, and they use this cursor position. If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COUT1.

Control Characters With COUT1 and BASICOUT

COUT1 and BASICOUT do not display control characters. Instead, the control characters listed in Tables 3-3a and 3-3b are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the appropriate escape code, as described in the section “Escape Codes With KEYIN” later in this chapter. The stop-list function, described separately, can only be invoked from the keyboard.

Table 3-3a. Control Characters With 80-Column Firmware Off

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.

Table 3-3b. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-K†	VT	clear EOS	Clears from cursor position to the end of the screen.
Control-L†	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.

Table 3-3b—Continued. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.
Control-N†	SO	normal	Sets display format normal.
Control-O†	SI	inverse	Sets display format inverse.
Control-Q†	DC1	40-column	Sets display to 40-column.
Control-R†	DC2	80-column	Sets display to 80-column.
Control-S*	DC3	stop-list	Stops listing characters on the display until another key is pressed.
Control-U †	NAK	quit	Deactivates 80-column video firmware.
Control-V †	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.
Control-W †	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.
Control-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.
Control-Y †	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).
Control-Z †	SUB	clear line	Clears the line the cursor position is on.
Control-[ESC	enable MouseText	Map inverse uppercase characters to MouseText characters.
Control-\†	FS	forward space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.
Control-]†	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window).
Control-__	US	up	Moves cursor up a line, no scroll.

* Only works from the keyboard.

† Doesn't work from the keyboard.

The Stop-List Feature

When you are using any program that displays text via COUT1 (or BASICOUT), you can make it stop updating the display by holding down **CONTROL** and pressing **S**. Whenever COUT1 gets a carriage return from the program, it checks to see if you have pressed **CONTROL-S**. If you have, COUT1 stops and waits for you to press another key. When you want COUT1 to resume, press another key; COUT1 will send the carriage return it got earlier to the display, then continue normally. The character code of the key you pressed to resume displaying is ignored unless you pressed **CONTROL-C**. COUT1 passes Control-C back to the program; if it is a BASIC program, this enables you to terminate the program while in stop-list mode.

The Text Window

After starting up the computer or after a reset, the firmware uses the entire display. However, you can restrict video activity to any rectangular portion of the display you wish. The active portion of the display is called the text window. COUT1 or BASICOUT puts characters into the window only; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can set the top, bottom, left side, and width of the text window by storing the appropriate values into four locations in memory. This enables your programs to control the placement of text in the display and to protect other portions of the screen from being written over by new text.

Memory location 32 (hexadecimal \$20) contains the number of the leftmost column in the text window. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal \$27); in an 80-column display, the maximum value is 79 (hexadecimal \$4F).

Memory location 33 (hexadecimal \$21) holds the width of the text window. For a 40-column display, it is normally 40 (hexadecimal \$28); for an 80-column display, it is normally 80 (hexadecimal \$50).

Original Iie

COUT1 truncates the column width to an even value on the original Apple Iie.

▲Warning

On an original Apple IIe, be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40 or 80). If this happens, it is possible for COUT1 to put characters into memory locations outside the display page, possibly into your current program or data space.

Memory location 34 (hexadecimal \$22) contains the number of the top line of the text window. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal \$17).

Memory location 35 (hexadecimal \$23) contains the number of the bottom line of the screen, plus 1. It is normally 24 (hexadecimal \$18) for the bottom line of the display. Its minimum value is 1.

After you have changed the text window boundaries, nothing is affected until you send a character to the screen.

▲Warning

Any time you change the boundaries of the text window, you should make sure that the current cursor position (stored at CH and CV) is inside the new window. If it is outside, it is possible for COUT1 to put characters into memory locations outside the display page, possibly destroying programs or data.

Table 3-4 summarizes the memory locations and the possible values for the window parameters.

Table 3-4. Text Window Memory Locations

Window Parameter	Location		Minimum Value		Normal Values				Maximum Values			
					40 col.		80 col.		40 col.		80 col.	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left Edge	32	\$20	00	\$00	00	\$00	00	\$00	39	\$27	79	\$4F
Width	33	\$21	00	\$00	40	\$28	80	\$50	40	\$28	80	\$50
Top Edge	34	\$22	00	\$00	00	\$00	00	\$00	23	\$17	23	\$17
Bottom Edge	35	\$23	01	\$01	24	\$18	24	\$18	24	\$18	24	\$18

Inverse and Flashing Text

Subroutine COUT1 can display text in normal format, inverse format, or, with some restrictions, flashing format. The display format for any character in the display depends on two things: the character set being used at the moment, and the setting of the two high-order bits of the character's byte in the display memory.

As it sends your text characters to the display, COUT1 sets the high-order bits according to the value stored at memory location 50 (hexadecimal \$32). If that value is 255 (hexadecimal \$FF), COUT1 sets the characters to display in normal format; if the value is 63 (hexadecimal \$3F), COUT1 sets the characters to inverse format. If the value is 127 (hexadecimal \$7F) and if you have selected the primary character set, the characters will be displayed in flashing format. Note that flashing format is not available in the alternate character set.

Table 3-5. Text Format Control Values

Note: These mask values apply only to the primary character set (see text).

Mask Value		Display Format
Dec	Hex	
255	\$FF	Normal, uppercase, and lowercase
127	\$7F	Flashing, uppercase, and symbols
63	\$3F	Inverse, uppercase, and lowercase

To control the display format of the characters, routine COUT1 uses the value at location 50 as a logical mask to force the setting of the two high-order bits of each character byte it puts into the display page. It does this by performing the logical AND function on the data byte and the mask byte. The result byte contains a 0 in any bit that was 0 in the mask. BASICOUT, used when the 80-column firmware is active, changes only the high-order bit of the data.

Important!

If the 80-column firmware is inactive and you store a mask value at location 50 with zeros in its low-order bits, COUT1 will mask out those bits in your text. As a result, some characters will be transformed into other characters. You should set the mask to the values given in Table 3-5 only.

Switching between character sets is described in the section “Display Mode Switching” in Chapter 2.

If you set the mask value at location 50 to 127 (hexadecimal \$7F), the high-order bit of each result byte will be 0, and the characters will be displayed either as lowercase or as flashing, depending on which character set you have selected. Refer to the tables of display character sets in Chapter 2. In the primary character set, the next-highest bit, bit 6, selects flashing format with uppercase characters. With the primary character set you can display lowercase characters in normal format and uppercase characters in normal, inverse, and flashing formats. In the alternate character set, bit 6 selects lowercase or special characters. With the alternate character set you can display uppercase and lowercase characters in normal and inverse formats.

Original Iie

On the original Apple Iie, the MouseText characters are replaced by uppercase inverse characters.

Standard Input Features

For more information on GETLN, see the section “Editing With GETLN,” later in this chapter.

The Apple Iie’s firmware includes two different subroutines for reading from the keyboard. One subroutine is named RDKEY, which stands for *read key*. It calls the standard character input subroutine KEYIN (or BASICIN when the 80-column firmware is active) which accepts one character at a time from the keyboard.

The other subroutine is named GETLN, which stands for *get line*. By making repeated calls to RDKEY, GETLN accepts a sequence of characters terminated with a carriage return. GETLN also provides on-screen editing features.

RDKEY Input Subroutine

A program gets a character from the keyboard by making a subroutine call to RDKEY at memory location \$FD0C. RDKEY sets the character at the cursor position to flash, then passes control via the input link KSW to the current input subroutine, which is normally KEYIN or BASICIN.

RDKEY displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COUT routine, described earlier). The cursor displayed by RDKEY is a flashing version of whatever character happens to be at that position on the screen. It is usually a space, so the cursor appears as a blinking rectangle.

KEYIN Input Subroutine

KEYIN is the standard input subroutine when the 80-column firmware is inactive; BASICIN is used when the 80-column firmware is active. When called, the subroutine waits until the user presses a key, then returns with the key code in the accumulator.

If the 80-column firmware is inactive, KEYIN displays a cursor by alternately storing a checkerboard block in the cursor location, then storing the original character, then the checkerboard again. If the firmware is active, BASICIN displays a steady inverse space (rectangle), unless you are in escape mode, when it displays a plus sign (+) in inverse format.

KEYIN also generates a random number. While it is waiting for the user to press a key, KEYIN repeatedly increments the 16-bit number in memory locations 78 and 79 (hexadecimal \$4E and \$4F). This number keeps increasing from 0 to 65535, then starts over again at 0. The value of this number changes so rapidly that there is no way to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a random number routine.

When the user presses a key, KEYIN accepts the character, stops displaying the cursor, and returns to the calling program with the character in the accumulator.

Escape Codes

KEYIN has special functions that you invoke by typing escape codes on the keyboard. An escape code is obtained by pressing **[ESC]**, releasing it, and then pressing some other key. See Table 3-6; the notation in the table means press **[ESC]**, release it, then press the key that follows.

Table 3-6 includes three sets of cursor-control keys. The first set consists of **[ESC]** followed by A, B, C, or D. The letter keys can be either uppercase or lowercase. These keys are the standard cursor-motion keys on older Apple II models; they are present on the Apple IIe primarily for compatibility with programs written for old machines.

Cursor Motion in Escape Mode

The second and third set of cursor-control keys are listed together because they activate escape mode. In escape mode, you can keep using the cursor-motion keys without pressing **[ESC]** again. This enables you to perform repeated cursor moves by holding down the appropriate key.

Escape mode is described in the next section, "Escape Codes."

When the 80-column firmware is active, you can tell when BASICIN is in escape mode: it displays a plus sign in inverse format as the cursor. You leave escape mode by typing any key other than a cursor-motion key.

The escape codes with the directional arrow keys are the standard cursor-motion keys on the Apple IIe. The escape codes with the I, J, K, and M keys are the standard cursor-motion keys on the Apple II Plus, and are present on the Apple IIe for compatibility with the Apple II Plus. On the Apple IIe, the escape codes with the I, J, K, and M keys function with either uppercase or lowercase letters.

Table 3-6. Escape Codes

Escape Code	Function
ESC @	Clears window and homes cursor (places it in upper-left corner of screen), then exits from escape mode.
ESC A or a	Moves cursor right one line; exits from escape mode.
ESC B or b	Moves cursor left one line; exits from escape mode.
ESC C or c	Moves cursor down one line; exits from escape mode.
ESC D or d	Moves cursor up one line; exits from escape mode.
ESC E or e	Clears to end of line; exits from escape mode.
ESC F or f	Clears to bottom of window; exits from escape mode.
ESC I or i or ESC ↑	Moves the cursor up one line; remains in escape mode. See text.
ESC J or j or ESC ←	Moves the cursor left one space; remains in escape mode. See text.
ESC K or k or ESC →	Moves the cursor right one space; remains in escape mode. See text.
ESC M or m or ESC ↓	Moves the cursor down one line; remains in escape mode. See text.
ESC 4	If 80-column firmware is active, switches to 40-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
ESC 8	If 80-column firmware is active, switches to 80-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
ESC CONTROL D	Disables control characters; only carriage return, line feed, BELL, and backspace have an effect when printed.
ESC CONTROL E	Reactivates control characters.
ESC CONTROL Q	If 80-column firmware is active, deactivates 80-column firmware; sets links to KEYIN and COUT1; restores normal window size; exits from escape mode.

GETLN Input Subroutine

Programs often need strings of characters as input. While it is possible to call RDKEY repeatedly to get several characters from the keyboard, there is a more powerful subroutine you can use. This routine is named GETLN, which stands for *get line*, and starts at location \$FD6A. Using repeated calls to RDKEY, GETLN accepts characters from the standard input subroutine—usually KEYIN—and puts them into the input buffer located in the memory page from \$200 to \$2FF. GETLN also provides the user with on-screen editing and control features, described in the next section “Editing With GETLN.”

The first thing GETLN does when you call it is display a prompting character, called simply a **prompt**. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. For example, an INPUT statement in a BASIC program displays a question mark (?) as a prompt. The prompt characters used by the different programs on the Apple IIe are shown in Table 3-7.

GETLN uses the character stored at memory location 51 (hexadecimal \$33) as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect, because both BASIC interpreters and the Monitor restore it each time they request input from the user.

Table 3-7. Prompt Characters

Prompt Character	Program Requesting Input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 5)

As you type the character string, GETLN sends each character to the standard output routine—normally COUT1—which displays it at the previous cursor position and puts the cursor at the next available position on the display, usually immediately to the right. As the cursor travels across the display, it indicates the position where the next character will be displayed.

GETLN stores the characters in its buffer, starting at memory location \$200 and using the X register to index the buffer. GETLN continues to accept and display characters until you press **RETURN**; then it clears the remainder of the line the cursor is on, stores the carriage-return code in the buffer, sends the carriage-return code to the display, and returns to the calling program.

The maximum line-length that GETLN can handle is 255 characters. If the user types more than this, GETLN sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GETLN sounds a bell (tone) at every keypress after the 248th.

Important!

In the Apple II and the Apple II Plus, the GETLN routine converts all input to uppercase. GETLN in the Apple IIe does not do this, even in Apple II mode. To get uppercase input for BASIC, use **CAPS LOCK**.

Editing With GETLN

Subroutine GETLN provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. For an introduction to editing with these features, refer to the *Applesoft Tutorial*. Any program that uses GETLN for reading the keyboard has these features.



Cancel Line

Any time you are typing a line, pressing **CONTROL-X** causes GETLN to cancel the line. GETLN displays a backslash (\) and issues a carriage return, then displays the prompt and waits for you to type a new line. GETLN takes the same action when you type more than 255 characters, as described earlier.

Backspace

When you press **←**, GETLN moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COUT, which moves the display position and the cursor back one space. If you type another character now, it will replace the character you backspaced over, both on the display and in the line buffer. Each time you press **←**, it moves the cursor left and deletes another character, until you reach the beginning of the line. If you then press **←** one more time, you have cancelled the line, and GETLN issues a carriage return and displays the prompt.

Retype

 has a function complementary to the backspace function. When you press , GETLN picks up the character at the display position just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you have just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display. (See the earlier section “Cursor Motion in Escape Mode.”)

Monitor Firmware Support

Table 3-8 summarizes the addresses and functions of the video display support routines the Monitor provides. These routines are described in the subsections that follow.

Table 3-8. Video Firmware Routines

Location	Name	Description
\$C307	BASICOUT	Displays a character on the screen when 80-column firmware is active.
\$FC9C	CLREOL	Clears to end of line from current cursor position.
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position.
\$FC42	CLREOP	Clears to bottom of window.
\$F832	CLRSCR	Clears the low-resolution screen.
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen.
\$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3).
\$FDF0	COUT1	Displays a character on the screen (Chapter 3).
\$FD8E	CROUT	Generates a carriage return character.
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character.
\$F819	HLINE	Draws a horizontal line of blocks.

Table 3-8—Continued. Video Firmware Routines

Location	Name	Description
\$FC58	HOME	Clears the window and puts cursor in upper-left corner of window.
\$F800	PLOT	Plots a single low-resolution block on the screen.
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device whose address is in CSW.
\$FDDA	PRBYTE	Prints a hexadecimal byte.
\$FF2D	PRERR	Sends ERR and Control-G to the output device whose output routine address is in CSW.
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number.
\$F941	PRNTAX	Prints contents of A and X in hexadecimal.
\$F871	SCRN	Reads color value of a low-resolution block on the screen.
\$F864	SETCOL	Sets the color for plotting in low-resolution.
\$FC24	VTABZ	Sets cursor vertical position. (Setting CV at location \$25 does not change vertical position until a carriage return.)
\$F828	VLINE	Draws a vertical line of low-resolution blocks.

BASICOUT, \$C307

BASICOUT is essentially the same as COUT1—BASICOUT is used instead of COUT1 when the 80-column firmware is active. BASICOUT displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). BASICOUT handles control characters; see Table 3-3b. When it returns control to the calling program, all registers are intact.

CLREOL, \$FC9C

CLREOL clears a text line from the cursor position to the right edge of the window. This routine destroys the contents of A and Y.

CLEOLZ, \$FC9E

CLEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL, which is indexed by the contents of the Y register. This routine destroys the contents of A and Y.

CLREOP, \$FC42

CLREOP clears the text window from the cursor position to the bottom of the window. This routine destroys the contents of A and Y.

CLRSCR, \$F832

CLRSCR clears the low-resolution graphics display to black. If you call this routine while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. This routine destroys the contents of A and Y.

CLRTOP, \$F836

CLRTOP is the same as CLRSCR, except that it clears only the top 40 rows of the low-resolution display.

COUT, \$FDED

COUT calls the current character output subroutine. (See the section “COUT Output Subroutine” earlier in this chapter.) The character to be sent to the output device should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations \$36 and \$37), which is usually the standard character output subroutine COUT1 (or BASICOUT).

COUT1, \$FDF0

COUT1 displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). It handles these control characters: carriage return, line feed, backspace, and bell. When it returns control to the calling program, all registers are intact.

CROUT, \$FD8E

CROUT sends a carriage return to the current output device.

See the section “Control Characters With COUT1 and BASICOUT,” earlier in this chapter for more information on COUT1.

CROUT1, \$FD8B

CROUT1 clears the screen from the current cursor position to the edge of the text window, then calls CROUT.

HLINE, \$F819

HLINE draws a horizontal line of blocks of the color set by SETCOL on the low-resolution graphics display. Call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLINE returns with A and Y scrambled and X intact.

HOME, \$FC58

HOME clears the display and puts the cursor in the upper-left corner of the screen.

PLOT, \$F800

PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. Call PLOT with the vertical coordinate of the line in the accumulator, and its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

PRBL2, \$F94A

PRBL2 sends from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to send. If X = \$00, then PRBLANK will send 256 blanks.

PRBYTE, \$FDDA

PRBYTE sends the contents of the accumulator in hexadecimal to the current output device. The contents of the accumulator are scrambled.

PRERR, \$FF2D

PRERR sends the word **ERR**, followed by a bell character, to the standard output device. On return, the accumulator is scrambled.

PRHEX, \$FDE3

PRHEX prints the lower nibble of the byte in the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.

PRNTAX, \$F941

PRTAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte printed, and the X register contains the second. On return, the contents of the accumulator are scrambled.

SCRN, \$F871

SCRN returns the color value of a single block on the low-resolution display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. The block's color is returned in the accumulator. No other registers are changed.

SETCOL, \$F864

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 2-6.

VTABZ, \$FC24

VTABZ sets the cursor vertical position. Unlike setting the position at location \$25, change of cursor position doesn't wait until a carriage return character has been sent.

VLINE, \$F828

VLINE draws a vertical line of blocks of the color set by SETCOL on the low-resolution display. Call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLINE returns with the accumulator scrambled.

I/O Firmware Support

Apple IIe video firmware conforms to the I/O firmware protocol of Apple II Pascal 1.1. However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode. The video protocol table is shown in Table 3-9.

Table 3-9. Slot 3 Firmware Protocol Table

Address	Value	Description
\$C30B	\$01	Generic signature byte of firmware cards
\$C30C	\$88	80-column card device signature
\$C30D	\$ii	\$C3ii is entry point of initialization routine (PINIT).
\$C30E	\$rr	\$C3rr is entry point of read routine (PREAD).
\$C30F	\$ww	\$C3ww is entry point of write routine (PWRITE).
\$C310	\$ss	\$C3ss is entry point of the status routine (PSTATUS).

PINIT, \$C30D

PINIT does the following:

- Sets a full 80-column window.
- Sets 80STORE (\$C001).
- Sets 80COL (\$C00D).
- Switches on ALTCHAR (\$C00F).
- Clears the screen; places cursor in upper-left corner.
- Displays the cursor.

PREAD, \$C30E

PREAD reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a zero in the X register to indicate IORESULT = GOOD.

PWRITE, \$C30F

PWRITE should be called after placing a character in the accumulator with its high bit cleared. PWRITE does the following:

- Turns the cursor off.
- If the character in the accumulator is not a control character, turns the high bit on for normal display or off for inverse display, displays it at the current cursor position, and advances the cursor; if at the end of a line, does carriage return but not line feed. (See Table 3-10 for control character functions.)

When PWRITE has completed this, it

- turns the cursor back on (if it was not intentionally turned off)
- puts a zero in the X register (IORESULT = GOOD) and returns to the calling program.

Table 3-10. Pascal Video Control Functions

Control-	Hex	Function Performed
E or e	\$05	Turns cursor on (enables cursor display).
F or f	\$06	Turns cursor off (disables cursor display).
G or g	\$07	Sounds bell (beeps).
H or h	\$08	Moves cursor left one column. If cursor was at beginning of line, moves it to end of previous line.
J or j	\$0A	Moves cursor down one row; scrolls if needed.
K or k	\$0B	Clears to end of screen.
L or l	\$0C	Clears screen; moves cursor to upper-left of screen.
M or m	\$0D	Moves cursor to column 0.
N or n	\$0E	Displays subsequent characters in normal video. (Characters already on display are unaffected.)

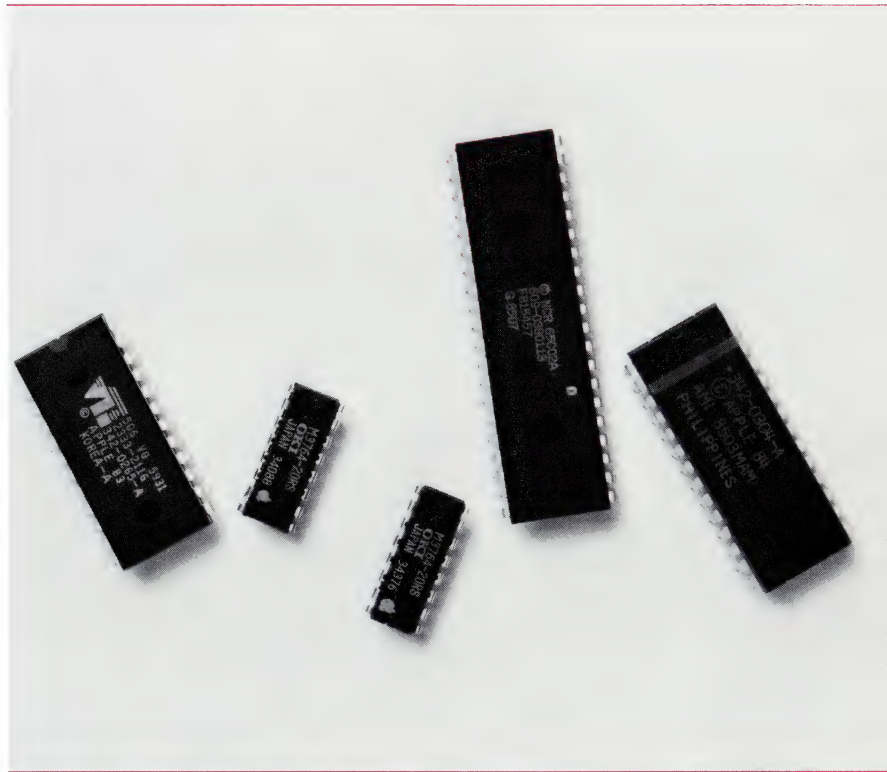
Table 3-10—Continued. Pascal Video Control Functions

Control-	Hex	Function Performed
O or o	\$0F	Displays subsequent characters in inverse video. (Characters already on display are unaffected.)
V or v	\$16	Scrolls screen up one line; clears bottom line.
W or w	\$17	Scrolls screen down one line; clears top line.
Y or y	\$19	Moves cursor to upper-left (home) position on screen.
Z or z	\$1A	Clears entire line that cursor is on.
or \	\$1C	Moves cursor right one column; if at end of line, does Control-M.
} or }	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor.
^ or 6	\$1E	GOTOxy: initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively.
—	\$1F	If not at top of screen, moves cursor up one line.

PSTATUS, \$C310

A program that calls PSTATUS must first put a request code in the accumulator: either a 0, meaning “Ready for output?” or a 1, meaning “Is there any input?” PSTATUS returns with the reply in the carry bit: 0 (No) or 1 (Yes).

PSTATUS returns with a 0 in the X register (IORESULT = GOOD), unless the request was not 0 or 1; then PSTATUS returns with a 3 in the X register (IORESULT = ILLEGAL OPERATION).



The Apple IIe's microprocessor can address 65,536 (64K) memory locations. All of the programmable storage (RAM and ROM) and input and output devices are allocated locations in this 64K address space. Some functions share the same addresses—but not at the same time.

For information about these shared address spaces, see the section “Bank-Switched Memory” in this chapter and the sections “Other Uses of I/O Memory Space” and “Expansion ROM Space” in Chapter 6.

Original IIe

The original version of the Apple IIe, as well as the Apple II Plus and Apple II, use the 6502 microprocessor. The 6502 lacks ten instructions and two addressing modes found on the 65C02 of the enhanced Apple IIe, but is otherwise functionally similar. For more information about the differences between the two processors, see Appendix A. In this manual, unless otherwise stated, the two processors are effectively the same.

For details of the built-in I/O features, refer to the descriptions in Chapters 2 and 3.

For information about I/O operations with peripheral cards, refer to Chapter 6.

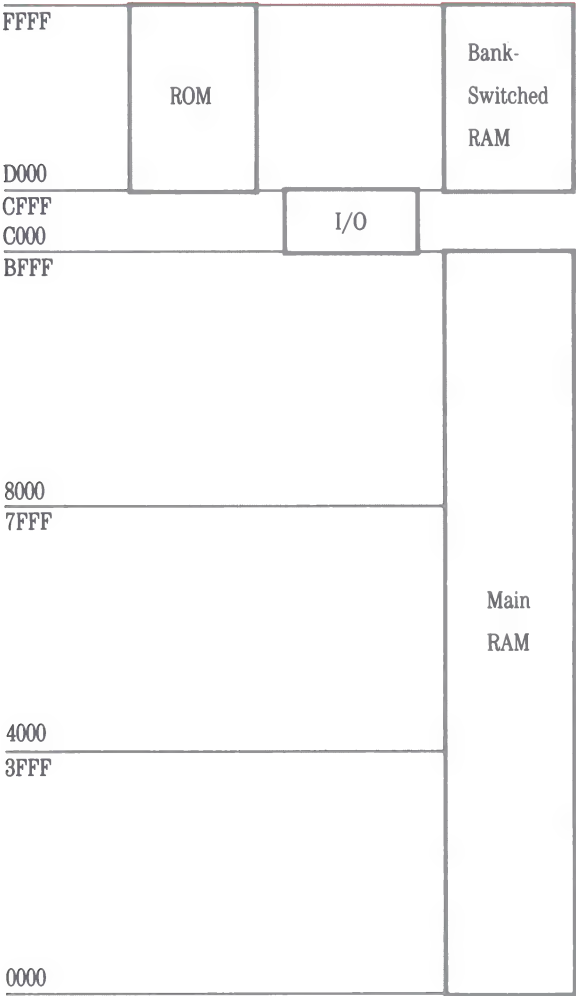
All input and output in the Apple IIe is memory mapped. This means that all devices connected to the Apple IIe appear to be memory locations to the computer. In this chapter, the I/O memory spaces are described simply as blocks of memory.

Programmers often refer to the Apple IIe's memory in 256-byte blocks called pages. One reason for this is that a one-byte address counter or index register can specify one of 256 different locations. Thus, page 0 consists of memory locations from 0 to 255 (hexadecimal \$00 to \$FF), inclusive. Page 1 consists of locations 256 to 511 (hexadecimal \$0100 to \$01FF); note that the page number is the high-order part of the hexadecimal address. Don't confuse this kind of page with the display buffers in the Apple IIe, which are sometimes referred to as Page 1 and Page 2.

Main Memory Map

The map of the main memory address space in Figure 4-1 shows the functions of the major areas of memory. For more details on the I/O space from 48K to 52K (\$C000 through \$CFFF), refer to Chapter 2 and Chapter 6; the bank-switched memory in the memory space from 52K to 64K (\$D000 through \$FFFF) is described in the section “Bank-Switched Memory” later in this chapter.

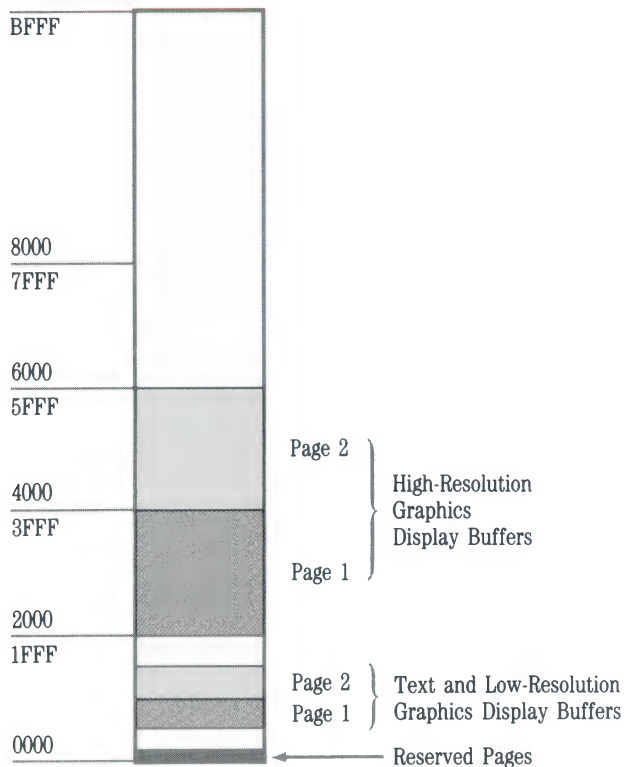
Figure 4-1. System Memory Map



RAM Memory Allocation

As Figure 4-1 shows, the major portion of the Apple IIe's memory space is allocated to programmable storage (RAM). Figure 4-2 shows the areas allocated to RAM. The main RAM memory extends from location 0 to location 49151 (hex \$BFFF), and occupies pages 0 through 191 (hexadecimal \$BF). There is also RAM storage in the bank-switched space from 53248 to 65535 (hexadecimal \$D000 to \$FFFF), described in the section "Bank-Switched Memory" later in this chapter, and auxiliary RAM, described in the section "Auxiliary Memory and Firmware" later in this chapter.

Figure 4-2. RAM Allocation Map



Reserved Memory Pages

Most of the Apple IIe's RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware and the BASIC interpreters. The reserved pages are described in the following sections.

Important!

The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain, or you will cause the system to malfunction.

Page Zero

Several of the 65C02 microprocessor's addressing modes require the use of addresses in page zero, also called zero page. The Monitor, the BASIC interpreters, DOS 3.3, and ProDOS all make extensive use of page zero.

To use indirect addressing in your assembly-language programs, you must store base addresses in page zero. At the same time, you must avoid interfering with the other programs that use page zero—the Monitor, the BASIC interpreters, and the disk operating systems. One way to avoid conflicts is to use only those page-zero locations not already used by other programs. Tables 4-1 through 4-5 show the locations in page zero used by the Monitor, Applesoft BASIC, Integer BASIC, DOS 3.3, and ProDOS.

As you can see from the tables, page zero is pretty well used up, except for a few bytes here and there. It's hard to find more than one or two bytes that aren't used by either BASIC, ProDOS, the Monitor, or DOS. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: save the contents of part of page zero, use that part, then restore the previous contents before you pass control to another program.

The 65C02 Stack

The 65C02 microprocessor uses page 1 as the stack—the place where subroutine return addresses are stored, in last-in, first-out sequence. Many programs also use the stack for temporary storage of the registers (via push and pull operations). You can do the same, but you should use it sparingly. The stack pointer is eight bits long, so the stack can hold only 256 bytes of information at a time. When you store the 257th byte in the stack, the stack pointer repeats itself, or wraps around, so that the new byte replaces the first byte stored, which is now lost. This writing over old data is called stack overflow, and when it happens, the program continues to run normally until the lost information is needed, whereupon the program terminates catastrophically.

The Input Buffer

The GETLN input routine, which is used by the Monitor and the BASIC interpreters, uses page 2 as its keyboard-input buffer. The size of this buffer sets the maximum size of input strings. (Note: Applesoft uses only the first 237 bytes, although it permits you to type in 256 characters.) If you know that you won't be typing any long input strings, you can store temporary data at the upper end of page 2.

Link-Address Storage

For more information about links, see the section "Changing the Standard I/O Links" in Chapter 6.

The Monitor, ProDOS, and DOS 3.3 all use the upper part of page 3 for link addresses or vectors.

BASIC programs sometimes need short machine-language routines. These routines are usually stored in the lower part of page 3.

The Display Buffers

See Chapter 6 for information on the memory locations that are reserved for peripheral cards.

The primary text and low-resolution-graphics display buffer occupies memory pages 4 through 7 (locations 1024 through 2047, hexadecimal \$0400 through \$07FF). This entire 1024-byte area is called text Page 1, and it is not usable for program and data storage. There are 64 locations in this area that are not displayed on the screen; these locations are reserved for use by the peripheral cards.

Text Page 2, the alternate text and low-resolution-graphics display buffer, occupies memory pages 8 through 11 (locations 2048 through 3071, hexadecimal \$0800 through \$0BFF). Most programs do not use Page 2 for displays, so they can use this area for program or data storage.

The primary high-resolution-graphics display buffer, called high-resolution Page 1, occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal \$2000 through \$3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution Page 2 occupies memory pages 64 through 95 (locations 16384 through 24575, hexadecimal \$4000 through \$5FFF). Most programs use this area for program or data storage.

For more information about the display buffers, see the section "Video Display Pages" in Chapter 2.

The primary double-high-resolution-graphics display buffer, called double-high-resolution Page 1, occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal \$2000 through \$3FFF) in both main and auxiliary memory. If your program doesn't use high-resolution or double-high-resolution graphics, this area of main memory is usable for programs or data.

Table 4-1. Monitor Zero-Page Use

High Nibble of Address	Low Nibble of Address															
	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$00																
\$10																•*
\$20		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$30		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$40		•	•	•	•	•	•	•	•	•					•	•
\$50		•	•	•	•	•	•									
\$60																
\$70																
\$80																
\$90																
\$A0																
\$B0																
\$C0																
\$D0																
\$E0																
\$F0																

* Byte used in original Apple IIe ROMs, now free.

Table 4-2. Applesoft Zero-Page Use

High Nibble of Address	Low Nibble of Address															
	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$00		•	•	•	•	•	•				•	•	•	•	•	•
\$10		•	•	•	•	•	•	•	•	•	•	•	•	•		
\$20							•	•					•	•		•
\$30		•		•									•	•	•	•
\$40																
\$50		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$60		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$70		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$80		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$90		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$A0		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$B0		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
\$C0		•	•	•	•	•	•	•	•	•	•	•	•	•		
\$D0		•	•	•	•	•	•		•	•	•	•	•	•	•	•
\$E0		•	•	•	•	•	•	•	•	•	•					
\$F0		•	•	•	•	•	•	•	•	•						•

Table 4-3. Integer BASIC Zero-Page Use

High Nibble of Address	Low Nibble of Address															
	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$00																
\$10																
\$20																
\$30																
\$40																
\$50																
\$60																
\$70																
\$80																
\$90																
\$A0																
\$B0																
\$C0																
\$D0																
\$E0																
\$F0																

Table 4-4. DOS 3.3 Zero-Page Use

High Nibble of Address	Low Nibble of Address															
	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$00																
\$10																
\$20																
\$30																
\$40																
\$50																
\$60																
\$70																
\$80																
\$90																
\$A0																
\$B0																
\$C0																
\$D0																
\$E0																
\$F0																

Table 4-5. ProDOS MLI and Disk-Driver Zero-Page Use

High Nibble of Address	Low Nibble of Address															
	\$0	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	\$A	\$B	\$C	\$D	\$E	\$F
\$00	•	•														
\$10																
\$20																
\$30											•	•	•	•	•	•
\$40	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
\$50																
\$60																
\$70																
\$80																
\$90																
\$A0																
\$B0																
\$C0																
\$D0																
\$E0																
\$F0																

Bank-Switched Memory

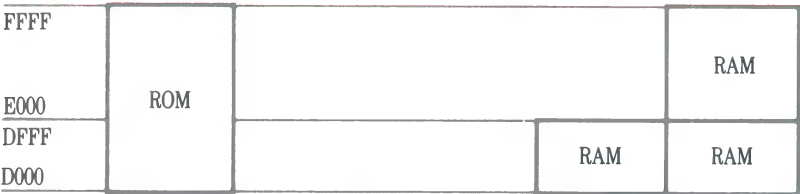
The memory address space from 52K to 64K (hexadecimal \$D000 through \$FFFF) is doubly allocated: it is used for both ROM and RAM. The 12K bytes of ROM (read-only memory) in this address space contain the Monitor and the Applesoft BASIC interpreter. Alternatively, there are 16K bytes of RAM in this space. The RAM is normally used for storing either the Integer BASIC interpreter or part of the Pascal Operating System (purchased separately).

You may be wondering why this part of memory has such a split personality. Some of the reasons are historical: the Apple IIe is able to run software written for the Apple II and Apple II Plus because it uses this part of memory in the same way they do. It is convenient to have the Applesoft interpreter in ROM, but the Apple IIe, like an Apple II with a language card, is also able to use that address space for other things when Applesoft is not needed.

You may also be wondering how 16K bytes of RAM is mapped into only 12K bytes of address space. The usual answer is that it's done with mirrors, and that isn't a bad analogy: the 4K-byte address space from 52K to 56K (hexadecimal \$D000 through \$DFFF) is used twice.

Switching different blocks of memory into the same address space is called bank switching. There are actually two examples of bank switching going on here: first, the entire address space from 52K to 64K (\$D000 through \$FFFF) is switched between ROM and RAM, and second, the address space from 52K to 56K (\$D000 to \$DFFF) is switched between two different blocks of RAM.

Figure 4-3. Bank-Switched Memory Map



Setting Bank Switches

You switch banks of memory in the same way you switch other functions in the Apple IIe: by using soft switches. Read operations to these soft switches do three things: select either RAM or ROM in this memory space; enable or inhibit writing to the RAM (write-protect); and select the first or second 4K-byte bank of RAM in the address space \$D000 to \$DFFF.

▲Warning

Do not use these switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program.

Table 4-6 shows the addresses of the soft switches for enabling all combinations of reading and writing in this memory space. All of the hexadecimal values of the addresses are of the form \$C08x. Notice that several addresses perform the same function: this is because the functions are activated by single address bits. For example, any address of the form \$C08x with a 1 in the low-order bit enables the RAM for writing. Similarly, bit 3 of the address selects which 4K block of RAM to use for the address space \$D000-\$DFFF; if bit 3 is 0, the first bank of RAM is used, and if bit 3 is 1, the second bank is used.

When RAM is not enabled for reading, the ROM in this address space is enabled. Even when RAM is not enabled for reading, it can still be written to if it is write-enabled.

When you turn power on or reset the Apple IIe, it initializes the bank switches for reading the ROM and writing the RAM, using the second bank of RAM. Note that this is different from the reset on the Apple II Plus, which didn't affect the bank-switched memory (the language card). On the Apple IIe, you can't use the reset vector to return control to a program in bank-switched memory, as you could on the Apple II Plus.

Reset With Integer BASIC: When you are using Integer BASIC on the Apple IIe, reset works correctly, restarting BASIC with your program intact. This happens because the reset vector transfers control to DOS, and DOS resets the switches for the current version of BASIC.

Table 4-6. Bank Select Switches

Note: R means read the location, W means write anything to the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
	R	\$C080	Read RAM; no write; use \$D000 bank 2.
	RR	\$C081	Read ROM; write RAM; use \$D000 bank 2.
	R	\$C082	Read ROM; no write; use \$D000 bank 2.
	RR	\$C083	Read and write RAM; use \$D000 bank 2.
	R	\$C088	Read RAM; no write; use \$D000 bank 1.
	RR	\$C089	Read ROM; write RAM; use \$D000 bank 1.
	R	\$C08A	Read ROM; no write; use \$D000 bank 1.
	RR	\$C08B	Read and write RAM; use \$D000 bank 1.
RDBNK2	R7	\$C011	Read whether \$D000 bank 2 (1) or bank 1 (0).
RDLGRAM	R7	\$C012	Reading RAM (1) or ROM (0).
ALTZP	W	\$C008	Off: use main bank, page 0 and page 1.
ALTZP	W	\$C009	On: use auxiliary bank, page 0 and page 1.
RDALTZP	R7	\$C016	Read whether auxiliary (1) or main (0) bank.

Reading and Writing to RAM Banks: Note that you can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well.

Reading RAM and ROM: You can't read from ROM in part of the bank-switched memory and read from RAM in the rest: specifically, you can't read the Monitor in ROM while reading bank-switched RAM. If you want to use the Monitor firmware with a program in bank-switched RAM, copy the Monitor from ROM (locations \$F800 through \$FFCB) into bank-switched RAM. You can't do this from Pascal or ProDOS.

To see how to use these switches, look at the following section of an assembly-language program:

```

AD 83 C0      LDA $C083      *SELECT 2ND 4K BANK & READ/WRITE
AD 83 C0      LDA $C083      *BY TWO CONSECUTIVE READS
A9 D0         LDA #$D0       *SET UP...
85 01         STA BEGIN      *...NEW...
A9 FF         LDA #$FF       *...MAIN-MEMORY...
85 02         STA END        *...POINTERS...
20 97 C9      JSR RAMTST     *...FOR 12K BANK

AD 8B C0      LDA $C08B      *SELECT 1ST 4K BANK
20 97 C9      JSR RAMTST     *USE ABOVE POINTERS

AD 83 C0      LDA $C088      *SELECT 1ST BANK & WRITE PROTECT
A9 80         LDA #$80
E6 10         INC TSTNUM
20 58 C9      JSR WPTSINIT

AD 80 C0      LDA $C080      *SELECT 2ND BANK & WRITE PROTECT
E6 10         INC TSTNUM
A9 01         LDA #PAT12K
20 58 C9      JSR WPTSINIT

AD 8B C0      LDA $C08B      *SELECT 1ST BANK & READ/WRITE
AD 8B C0      LDA $C08B      *BY TWO CONSECUTIVE READS
E6 0E         INC RWMODE     *FLAG RAM IN READ/WRITE
E6 10         INC TSTNUM
A9 08         LDA #PAT4K
20 58 C9      JSR WPTSINIT

```

The LDA instruction, which performs a read operation to the specified memory location, is used for setting the soft switches. The unusual sequence of two consecutive LDA instructions performs the two consecutive reads that write-enable this area of RAM; in this case, the data that are read are not used.

Reading Bank Switches

You can read which language card bank is currently switched in by reading the soft switch at \$C011. You can find out whether the language card or ROM is switched in by reading \$C012. The only way that you can find out whether the language card RAM is write-enabled or not is by trying to write some data to the card's RAM space.

Auxiliary Memory and Firmware

By installing an optional card in the auxiliary slot, you can add more memory to the Apple IIe. One such card is the Apple IIe 80-Column Text Card, which has 1K bytes of additional RAM for expanding the text display from 40 columns to 80 columns.

Another optional card, the Apple IIe Extended 80-Column Text Card, has 64K of additional RAM. A 1K-byte area of this memory serves the same purpose as the memory on the 80-Column Text Card: expanding the text display to 80 columns. The other 63K bytes can be used as auxiliary program and data storage. If you use only 40-column displays, the entire 64K bytes is available for programs and data.

▲Warning

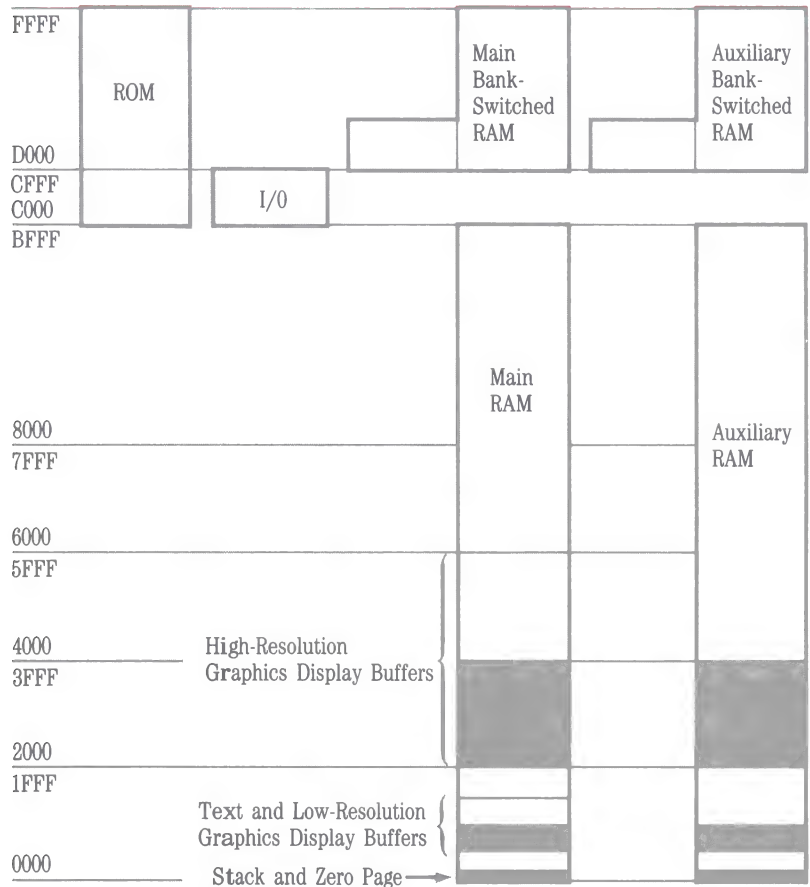
Do not attempt to use the auxiliary memory from a BASIC program. The BASIC interpreter uses several areas in main RAM, including the stack and the zero page. If you switch to auxiliary memory in these areas, the BASIC interpreter fails and you must reset the system and start over.

As you can see by studying the memory map in Figure 4-4, the auxiliary memory is broken into two large sections and one small one. The largest section is switched into the memory address space from 512 to 49151 (\$0200 through \$BFFF). This space includes the display buffer pages: as described in the section “Text Modes” in Chapter 2, space in auxiliary memory is used for one half of the 80-column text display. You can switch to the auxiliary memory for this entire memory space, or you can switch just the display pages: see the next section, “Memory Mode Switching.”

Soft Switches: If the only reason you are using auxiliary memory is for the 80-column display, note that you can store into the display page in auxiliary memory by using the 80STORE and PAGE2 soft switches described in the section “Display Mode Switching” in Chapter 2.

The other large section of auxiliary memory is switched into the memory address space from 52K to 64K (\$D000 through \$FFFF). This memory space and the switches that control it are described earlier in this chapter in the section “Bank-Switched Memory.” If you use the auxiliary RAM in this space, the soft switches have the same effect on the auxiliary RAM that they do on the main RAM: the bank switching is independent of the auxiliary-RAM switching.

Figure 4-4. Memory Map With Auxiliary Memory



Bank Switches: Note that the soft switches for the bank-switched memory, described in the previous section, do not change when you switch to auxiliary RAM. In particular, if ROM is enabled in the bank-switched memory space before you switch to auxiliary memory, the ROM will still be enabled after you switch. Any time you switch the bank-switched section of auxiliary memory in and out, you must also make sure that the bank switches are set properly.

When you switch in the auxiliary RAM in the bank-switched space, you also switch the first two pages, from 0 to 511 (\$0000 through \$01FF). This part of memory contains page zero, which is used for important data and base addresses, and page one, which is the 65C02 stack. The stack and zero page are switched this way so that system software running in the

bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K address space (from \$0200 to \$BFFF) in either main memory or auxiliary memory.

Memory Mode Switching

Switching the 48K section of memory is performed by two soft switches: the switch named RAMRD selects main or auxiliary memory for reading, and the one named RAMWRT selects main or auxiliary memory for writing. As shown in Table 4-7, each switch has a pair of memory locations dedicated to it, one to select main memory, and the other to select auxiliary memory. Enabling the read and write functions independently makes it possible for a program whose instructions are being fetched from one memory space to store data into the other memory space.

▲Warning

Do not use these switches without careful planning. Careless switching between main and auxiliary memories is almost certain to have catastrophic effects on the operation of the Apple IIe. For example, if you switch to auxiliary memory with no card in the slot, the program that is running will stop and you will have to reset the Apple IIe and start over.

Writing to the soft switch at location \$C003 turns RAMRD on and enables auxiliary memory for reading; writing to location \$C002 turns RAMRD off and enables main memory for reading. Writing to the soft switch at location \$C005 turns RAMWRT on and enables the auxiliary memory for writing; writing to location \$C004 turns RAMWRT off and enables main memory for writing. By setting these switches independently, you can use any of the four combinations of reading and writing in main or auxiliary memory.

Auxiliary memory corresponding to text Page 1 and high-resolution graphics Page 1 can be used as part of the address space from \$0200 to \$BFFF by using RAMRD and RAMWRT as described above. These areas in auxiliary RAM can also be controlled separately by using the switches described in the section “Display Mode Switching” in Chapter 2. Those switches are named 80STORE, PAGE2, and HIRES.

As shown in Table 4-7, the 80STORE switch functions as an enabling switch: with it on, the PAGE2 switch selects main memory or auxiliary memory. With the HIRES switch off, the memory space switched by PAGE2 is the text Page 1, from \$0400 to \$07FF; with HIRES on, PAGE2 switches both text Page 1 and high-resolution graphics Page 1, from \$2000 to \$3FFF.

If you are using both the auxiliary-RAM control switches and the auxiliary-display-page control switches, the display-page control switches take priority: if 80STORE is off, RAMRD and RAMWRT work for the entire

The next section, "Auxiliary-Memory Subroutines," describes firmware that you can call to help you switch between main and auxiliary memory.

memory space from \$0200 to \$BFFF, but if 80STORE is on, RAMRD and RAMWRT have no effect on the display page. Specifically, if 80STORE is on and HIRES is off, PAGE2 controls text Page 1 regardless of the settings of RAMRD and RAMWRT. Likewise, if 80STORE and HIRES are both on, PAGE2 controls both text Page 1 and high-resolution graphics Page 1, again regardless of RAMRD and RAMWRT.

A single soft switch named ALTZP (for alternate zero page) switches the bank-switched memory and the associated stack and zero page area between main and auxiliary memory. As shown in Table 4-7, writing to location \$C009 turns ALTZP on and selects auxiliary-memory stack and zero page; writing to the soft switch at location \$C008 turns ALTZP off and selects main-memory stack and zero page for both reading and writing.

Table 4-7. Auxiliary-Memory Select Switches.

Name	Function	Location			Notes
		Hex	Decimal		
RAMRD	Read auxiliary memory	\$C003	49155	-16381	Write
	Read main memory	\$C002	49154	-16382	Write
	Read RAMRD switch	\$C013	49171	-16365	Read
RAMWRT	Write auxiliary memory	\$C005	49157	-16379	Write
	Write main memory	\$C004	49156	-16380	Write
	Read RAMWRT switch	\$C014	49172	-16354	Read
80STORE	On: access display page	\$C001	49153	-16383	Write
	Off: use RAMRD, RAMWRT	\$C000	49152	-16384	Write
	Read 80STORE switch	\$C018	49176	-16360	Read
PAGE2	Page 2 on (aux. memory)	\$C055	49237	-16299	*
	Page 2 off (main memory)	\$C054	49236	-16300	*
	Read PAGE2 switch	\$C01C	49180	-16356	Read
HIRES	On: access high-res. pages	\$C057	49239	-16297	†
	Off: use RAMRD, RAMWRT	\$C056	49238	-16298	†
	Read HIRES switch	\$C01D	49181	-16355	Read
ALTZP	Auxiliary stack & z.p.	\$C009	49161	-16373	Write
	Main stack & zero page	\$C008	49160	-16374	Write
	Read ALTZP switch	\$C016	49174	-16352	Read

* When 80STORE is on, the PAGE2 switch selects main or auxiliary display memory.

† When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to switch between the high-resolution Page-1 area in main memory or auxiliary memory.

When these switches are on, auxiliary memory is being used; when they are off, main memory is being used.

There are three more locations associated with the auxiliary-memory switches. The high-order bits of the bytes you read at these locations tell you the settings of the three soft switches described above. The byte you read at location \$C013 has its high bit set to 1 if RAMRD is on (auxiliary memory is read-enabled), or 0 if RAMRD is off (the 48K block of main memory is read-enabled). The byte at location \$C014 has its high bit set to 1 if RAMWRT is on (auxiliary memory is write-enabled), or 0 if RAMWRT is off (the 48K block of main memory is write-enabled). The byte at location \$C016 has its high bit set to 1 if ALTZP is on (the bank-switched area, stack, and zero page in the auxiliary memory are selected), or 0 if ALTZP is off (these areas in main memory are selected).

Sharing Memory: In order to have enough memory locations for all of the soft switches and remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 4-7 share their memory locations with the keyboard functions listed in Table 2-2. The operations—read or write—shown in Table 4-7 for controlling the auxiliary memory are just the ones that are not used for reading the keyboard and clearing the strobe.

Auxiliary-Memory Subroutines

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in auxiliary-memory subroutines. These subroutines make it possible to use the auxiliary memory without having to manipulate the soft switches described in the previous section.

Important!

The subroutines described below make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program.

You use these built-in subroutines the same way you use the I/O subroutines described in Chapter 3: by making subroutine calls to their starting locations. Those locations are shown in Table 4-8.

Table 4-8. 48K RAM Transfer Routines

Name	Action	Hex	Function
AUXMOVE	JSR	\$C312	Moves data blocks between main and auxiliary 48K memory.
XFER	JMP	\$C314	Transfers program control between main and auxiliary 48K memory.

Moving Data to Auxiliary Memory

In your assembly-language programs, you can use the built-in subroutine named AUXMOVE to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page zero and set the carry bit to select the direction of the move—main to auxiliary or auxiliary to main.

▲Warning

Don't try to use AUXMOVE to copy data in page zero or page one (the 65C02 stack) or in the bank-switched memory (\$D000-\$FFFF). AUXMOVE uses page zero all during the copy, so it can't handle moves in the memory space switched by ALTZP.

The pairs of bytes you use for passing addresses to this subroutine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIe's built-in routines. The addresses of these byte pairs are shown in Table 4-9.

Table 4-9. Parameters for AUXMOVE Routine

Note: The X, Y, and A registers are preserved by AUXMOVE.

Name	Location	Parameter Passed
Carry		1 = Move from main to auxiliary memory 0 = Move from auxiliary to main memory
A1L	\$3C	Source starting address, low-order byte
A1H	\$3D	Source starting address, high-order byte
A2L	\$3E	Source ending address, low-order byte
A2H	\$3F	Source ending address, high-order byte
A4L	\$42	Destination starting address, low-order byte
A4H	\$43	Destination starting address, high-order byte

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

The AUXMOVE routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit; to copy data from auxiliary memory to main memory, clear the carry bit.

When you make the subroutine call to AUXMOVE, the subroutine copies the block of data as specified by the A byte pairs and the carry bit. When it is finished, the accumulator and the X and Y registers are just as they were when you called AUXMOVE.

Transferring Control to Auxiliary Memory

You can use the built-in routine named XFER to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFER: the address of the routine you are transferring to, the direction of the transfer (main to auxiliary or auxiliary to main), and which page zero and stack you want to use.

Table 4-10. Parameters for XFER Routine

Note: The X, Y, and A parameters are preserved by XFER.

Name or Location	Parameter Passed
Carry	1 = Transfer from main to auxiliary memory 0 = Transfer from auxiliary to main memory
Overflow	1 = Use page zero and stack in auxiliary memory 0 = Use page zero and stack in main memory
\$03ED	Program starting address, low-order byte
\$03EE	Program starting address, high-order byte

Put the transfer address into the two bytes at locations \$03ED and \$03EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory. Use the overflow bit to select which page zero and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit to use the auxiliary memory.

After you have set up the parameters, pass control to the XFER routine by a jump instruction, rather than a subroutine call. XFER saves the accumulator and the transfer address on the current stack, then sets up the soft switches for the parameters you have selected and jumps to the new program.

▲Warning

It is the programmer's responsibility to save the current stack pointer at \$0100 in main memory and the alternate stack pointer at \$0101 in auxiliary memory before calling XFER and to restore them after regaining control. Failure to do so will cause program errors.

The Reset Routine

To put the Apple IIe into a known state when it has just been turned on or after a program has malfunctioned, there is a procedure called the reset routine. The reset routine is built into the Apple IIe's firmware, and it is initiated any time you turn power on or press **RESET** while holding down **CONTROL**. The reset routine puts the Apple IIe into its normal operating mode and restarts the resident program.

When you initiate a reset, hardware in the Apple IIe sets the memory-controlling soft switches to normal: main board RAM and ROM are enabled, and, if there is an 80-column text card in the auxiliary slot, expansion slot 3 is allocated to the built-in 80-column firmware. Auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at \$D000.

The reset routine sets the display-controlling soft switches to display 40-column text Page 1 using the primary character set, then sets the window equal to the full 40-column display, puts the cursor at the bottom of the screen, and sets the display format to normal.

For information about the I/O links, see the section "Changing the Standard I/O Links" in Chapter 6.

For more information about peripheral-card ROM, see the section "Peripheral-Card ROM Space" in Chapter 6.

The reset routine sets the keyboard and display as the standard input and output devices by loading the standard I/O links. It turns annunciators 0 and 1 off and annunciators 2 and 3 on, clears the keyboard strobe, turns off any active peripheral-card ROM and outputs a bell (tone).

The Apple IIe has three types of reset: power-on reset, also called cold-start reset; warm-start reset; and forced cold-start reset. The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is valid or not, as described later in this chapter in the section "The Reset Vector." If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

The Cold-Start Procedure

If the reset vector is not valid, either the Apple IIe has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string **Apple IIe** (Apple II on an original IIe) at the top of the display. It loads the reset vector and the validity-check byte as described below, then starts checking the expansion slots to see if there is a disk drive controller card in one of them, starting with slot 7 and working down.

If it finds a controller card, it initiates the startup (bootstrap) routine that resides in the controller card's firmware. The startup routine then loads DOS or ProDOS from the disk in drive 1. When the operating system has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor just keeps spinning until you press

CONTROL-RESET.

If the reset routine doesn't find a controller card, or if you press **CONTROL-RESET** again before the startup procedure has been completed, the reset routine will continue without using the disk, and pass control to the built-in Applesoft interpreter.

The Warm-Start Procedure

Whenever you press **CONTROL-RESET** when the Apple IIe has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the resident program, which is normally the built-in Applesoft interpreter. If the resident program is indeed Applesoft, your Applesoft program and variables are still intact. If you are using DOS, it is the resident program and it restarts either Applesoft or Integer BASIC, whichever you were using when you pressed **CONTROL-RESET**.

Important!

A program in bank-switched RAM cannot use the reset vector to regain control after a reset, because the Apple IIe hardware enables ROM in the bank-switched memory space. If you are using Integer BASIC, which is in the bank-switched RAM, you are also using DOS, and it is DOS that controls the reset vector and restarts BASIC.

For more information about ProDOS and the startup procedure, see the *ProDOS Technical Reference Manual*.

Forced Cold Start

If a program has loaded the reset vector to point to the beginning of the program, as described in the next section, pressing **CONTROL-RESET** causes a warm-start reset that uses the vector to transfer control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down **⌘** and **CONTROL**, then pressing and releasing **RESET**.

Unconditional Restart: When you want to stop a program unconditionally—for example, to start up the Apple IIe with some other program—you should use the forced cold-start reset, **⌘-CONTROL-RESET**, instead of turning the power off and on.

Whenever you press **CONTROL-RESET**, firmware in the Apple IIe always checks to see whether either Apple key is down. If the **⌘** key is down, with or without the **⌘** key, the firmware performs the self-test described later in this chapter. If only the **⌘** key is down, the firmware starts a forced cold-start reset. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page 3 are the ones that contain the reset vector. The reset routine then performs a normal cold-start reset.

The Reset Vector

When you reset the Apple IIe, the reset routine transfers control to the resident program by means of an address stored in page 3 of main RAM. This address is called a vector because it directs program control to a specified destination. There are several other vector addresses stored in page 3, as shown in Table 4-11, including the interrupt vectors described in the section “Interrupts on the Enhanced Apple IIe” in Chapter 6, and the ProDOS and DOS vectors described in the *ProDOS Technical Reference Manual* and the *Apple II DOS Programmer’s Manual*.

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations 1010 and 1011 (hexadecimal \$03F2 and \$03F3). It then stores a validity-check byte, also called the power-up byte, at location 1012 (hexadecimal \$03F4). The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal \$A5). Each time you reset the Apple IIe, the reset routine uses this byte to determine whether the reset vector is still valid.

You can change the reset vector so that the reset routine will transfer control to your program instead of to the Applesoft interpreter. For this to work, you must also change the validity-check byte to the exclusive-OR of the high-order byte of your new reset vector with the constant 165 (\$A5). If you fail to do this, then the next time you reset the Apple IIe, the reset routine will determine that the reset vector is invalid and perform a cold-start reset, eventually transferring control to the disk startup routine or to Applesoft.


The reset routine has a subroutine that generates the validity-check byte for the current reset vector. You can use this subroutine by doing a subroutine call to location -1169 (hexadecimal \$FB6F). When your program finishes, it can return the Apple IIe to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.

Table 4-11. Page 3 Vectors

Vector Address	Vector Function
\$3F0 \$3F1	Address of the subroutine that handles BRK requests (normally \$59, \$FA).
\$3F2 \$3F3	Reset vector (see text).
\$3F4	Power-up byte (see text).
\$3F5 \$3F6 \$3F7	Jump instruction to the subroutine that handles Applesoft & commands (normally \$4C, \$58, \$FF).
\$3F8 \$3F9 \$3FA	Jump instruction to the subroutine that handles user CONTROL - Y commands.
\$3FB \$3FC \$3FD	Jump instruction to the subroutine that handles non-maskable interrupts.
\$3FE \$3FF	Interrupt vector (address of the subroutine that handles interrupt requests).

See “The User’s Interrupt Handler at \$3FE” in Chapter 6.

Automatic Self-Test

If you reset the Apple IIe by holding down  and **CONTROL** while pressing and releasing **RESET**, the reset routine will start running the built-in self-test. Successfully running this test assures you that the Apple IIe is operational.

▲Warning

The self-test routine tests the Apple IIe's programmable memory by writing and then reading it. All programs and data in programmable memory when you run the self-test are destroyed.

The self-test takes several seconds to run. The screen will display some patterns in low resolution mode which will change rapidly just before the self-test finishes. If the test finishes normally, the Apple IIe displays **System OK** and waits for you to restart the system.

If you have been running a program, some soft switches might be on when you run the self-test. If this happens, the self-test will display a message such as

IDU FLAG ES:1

Turn the power off for several seconds, then turn it back on and run the self-test again. If it still fails, there is really something wrong; to get it corrected, contact your authorized Apple dealer for service.



The starting addresses for all of the standard subroutines are listed in Appendix B.

The System Monitor is a set of subroutines in the Apple IIe firmware. The Monitor provides a standard interface to the built-in I/O devices described in Chapter 2. The I/O subroutines described in Chapter 3 are part of the System Monitor.

ProDOS, DOS 3.3, and the BASIC interpreters use these subroutines by direct calls to their starting locations, as described for the I/O subroutines in Chapter 3.

If you wish, you can call the standard subroutines from your programs in the same fashion.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor to

- ☐ look at one or more memory locations
- ☐ change the contents of any location
- ☐ write programs in machine language to be executed directly by the Apple IIe's microprocessor
- ☐ save blocks of data and programs onto cassette tape and read them back in again
- ☐ move and compare blocks of memory
- ☐ search for data bytes and ASCII characters in memory
- ☐ invoke other programs from the Monitor
- ☐ invoke the Mini-Assembler.

Invoking the Monitor

The System Monitor starts at memory location \$FF69 (decimal 65385 or -151). To invoke the Monitor, you make a CALL statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (*), appears on the left side of the display screen, followed by a blinking cursor.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing **CONTROL-RESET**, by pressing **CONTROL-C** then **RETURN**, or by typing **3D0G**, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location \$3D0.

Syntax of Monitor Commands

To give a command to the Monitor, you type a line on the keyboard, then press **RETURN**. The Monitor accepts the line using the standard I/O subroutine GETLN, described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return.

A Monitor command can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation. Hexadecimal notation uses the ten decimal digits (0-9) and the first six letters (A-F) to represent the sixteen values from 0 to 15. A pair of hexadecimal digits represent values from 0 to 255, corresponding to a byte, and a group of four hexadecimal digits can represent values from 0 to 65,536, corresponding to a word. Any address in the Apple IIe can be represented by four hexadecimal digits.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading zeros; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

See "Summary of Monitor Commands" at the end of this chapter.

Each command you type consists of one command character, usually the first letter of the command name. When the command is a letter, it can be either uppercase or lowercase. The Monitor recognizes 23 different command characters. Some of them are punctuation marks, some are letters, and some are control characters.

Note: Although the Monitor recognizes and interprets control characters typed on an input line, they do not appear on the screen.

This chapter contains many examples of the use of Monitor commands. In the examples, the commands and values you type are shown in a normal typeface and the responses of the Monitor are in a computer typeface. Of course, when you perform the examples, all of the characters that appear on the display screen will be in the same typeface. Some of the data values displayed by your Apple IIe may differ from the values printed in these examples, because they are variables stored in programmable memory.

Monitor Memory Commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the last opened location and the next changeable location.

Examining Memory Contents

When you type the address of a memory location and press **RETURN**, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

```
*E000  
E000- 20  
  
*33  
0033- AA  
*
```

Each time the Monitor displays the value stored at a location, it saves the address of that location as the last opened location and as the next changeable location.

Memory Dump

When you type a period (.) followed by an address, and then press **RETURN**, the Monitor displays a memory dump: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger than the last opened location the address after the period is.

```

*20
0020- 00

*.2B
0021- 28 00 18 0F 0C 00 00
0028- A8 06 D0 07

*300
0300- 99

*.315
0301- B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03

*.32A
0316- 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20
*
```

When the Monitor performs a memory dump, it starts at the location immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a multiple of eight—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by simply concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a memory range.

```
*300.32F
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 C8 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
```

```
*30.40
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
```

```
*E015.E025
E016- 4C ED FD
E018- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing **RETURN** by itself causes the Monitor to display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as the last opened location and the next changeable location.

```
*5
0005- 00
*RETURN
00 00

*RETURN
0008- 00 00 00 00 00 00 00 00
*32
0032- FF

*RETURN
AA 00 C2 05 C2
*RETURN
0038- 1B FD D0 03 3C 00 3F 00
*
```

Changing Memory Contents

The previous section showed you how to display the values stored in the Apple IIe's memory; this section shows you how to change those values. You can change any location in RAM—programmable memory—and you can also change the soft switches and output devices by changing the locations assigned to them.

▲Warning

Use these commands carefully. If you change the zero-page locations used by Applesoft, ProDOS, or DOS, you may lose programs or data stored in memory.

Changing One Byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon (:) followed by a value.

```
*0
```

```
0000- 00
```

```
*:5F
```

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

```
*0
```

```
0000- 5F
```

```
*
```

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the example, you type the address again to verify the change.

```
*302:42
```

```
*302
```

```
0302- 42
```

```
*
```

When you change the contents of a location, the value that was contained in that location disappears, never to be seen again. The new value will remain until you replace it with another value.

Changing Consecutive Locations

You don't have to type a separate command with an address, a colon, a value, and **RETURN** for each location you want to change. You can change the values of up to 85 consecutive locations at a time (or even more, if you omit leading zeros from the values) by typing only the initial address and colon followed by all the values separated by spaces, and ending with **RETURN**. The Monitor will duly store the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next changeable location. Thus, you can continue changing consecutive locations without typing an address on the next input line by typing another colon and more values. In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 0 3
```

```
*300
```

```
0300- 69
```

```
*RETURN
```

```
01 20 ED FD 4C 00 03
```

```
*10:0 1 2 3
```

```
*:4 5 6 7
```

```
*10.17
```

```
0010- 00 01 02 03 04 05 06 07
```

```
*
```

ASCII Input Mode

The enhanced Apple IIe has an ASCII input mode that lets you enter ASCII characters just as you can their hexadecimal ASCII equivalents by preceding the literal character with an apostrophe ('). This means that 'A is the same as \$C1 and 'B is the same as \$C2 to the Monitor. The ASCII value for *any* character following an apostrophe is used by the Monitor.

Each character to be placed in memory should be delimited by a leading apostrophe (') and a trailing space. The only exception to this rule is that the last character in the line is followed with a return character instead of a space. The following example would enter the string "Hooray for sushi!" at \$0300 in memory.

```
*300:'H'o'o'r'a'y' 'f'o'r' 's'u's'h'i'!
```

Important!

ASCII input mode sets the high bit of the code for a character that you enter. So 'A' will equal \$C1, not \$41.

Original IIfx

The original Apple IIfx does not have an ASCII input mode.

Moving Data in Memory

You can copy a block of data stored in a range of memory locations from one area in memory to another by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory (the source locations) and where you want the copy to go (the destination locations). You give this information to the Monitor by means of three addresses: the address of the first location in the destination and the addresses of the first and last locations in the source. You specify the starting and ending addresses of the source range by separating them with a period. You separate the destination address from the range addresses with a less-than character (<), which you may think of as an arrow pointing in the direction of the move. Finally, you tell the Monitor that this is a MOVE command by typing the letter M (in either lowercase or uppercase). The format of the complete MOVE command looks like this:

```
{destination} < {start} . {end} M
```

When you type the actual command, the words in braces should be replaced by hexadecimal addresses, and the braces and spaces should be omitted.

Here are some examples of Monitor commands, including some memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory; the actual MOVE commands end with the letter M.

*0.F

0000- 5F 00 05 07 00 00 00 00

0008- 00 00 00 00 00 00 00 00

*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00 03

*300.30C

0300- A9 8D 20 ED FD A9 45 20

0308- DA FD 4C 00 03

*0<300.30CM

*0.C

0000- A9 8D 20 ED FD A9 45 20

0008- DA FD 4C 00 03

*310<8.AM

*310.312

0310- DA FD 4C

*2<7.9M

*0.C

0000- A9 8D 20 DA FD A9 45 20

0008- DA FD 4C 00 03

*

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the last location in the source range as the last opened location, and the first location in the source range as the next changeable location. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

If the destination address of the MOVE command is inside the source range of addresses, then strange (and sometimes wonderful) things happen: the locations between the beginning of the source range and the destination address are treated as a sub-range and the values in this sub-range are replicated throughout the source range.

See the section “Special Tricks With the Monitor” later in this chapter for an interesting application of this feature.

Comparing Data in Memory

You can use the `VERIFY` command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the `VERIFY` command can be used immediately after a `MOVE` command to make sure that the move was successful.

The `VERIFY` command, like the `MOVE` command, needs a range and a destination. The syntax of the `VERIFY` command is

```
{destination} < {start} . {end} V
```

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to \$D, copy them to locations starting at \$300 with the `MOVE` command, and then compare them using the `VERIFY` command. When you use the `VERIFY` command after you change the value at location 6 to \$E4, it detects the change.

```
*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4 C5
```

```
*300<0.DM
```

```
*300<0.DV
```

```
*6:E4
```

```
*300<0.DV
```

```
0006-E4 (EE)
```

```
*
```

If the `VERIFY` command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, `VERIFY` displays nothing. The `VERIFY` command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the `MOVE` command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges will be compared. Like the `MOVE` command, the `VERIFY` command also does unusual things if the destination address is within the source range.

See the section "Special Tricks With the Monitor" later in this chapter.

Searching for Bytes in Memory

The SEARCH command lets you search for one or two bytes (either hexadecimal values or ASCII characters) in a range of memory. You must type in the ASCII string (or hexadecimal number or numbers) in reverse of the order that they appear in memory. Think of the SEARCH command as looking for items in a last-in, first-out queue.

The syntax of the SEARCH command is

```
{value or ASCII} < {start}|.end|S
```

If the byte (or two byte sequence) that you specify is in the specified memory range, the Monitor will return with a list of the addresses where that byte (or byte sequence) occurs. If the byte (or byte sequence) is not in the range, the Monitor just displays the prompt.

The following example looks for the character string *LO* in memory between \$0300 and \$03FF.

```
*O'L<300.3FFS
```

High Bit Set: Remember that ASCII input mode sets the high-order bit of each character that you enter.

The next example searches for the two-byte sequence \$FF11.

```
*11FF<300.3FFS
```

You can't search for a two-byte sequence with a high byte of 0. The Monitor ignores the high byte and searches for the low byte only. The sequence 00FF is seen by the Monitor SEARCH command as FF.

Original ILe

The Monitor in the original Apple ILe does not recognize the SEARCH command.

Examining and Changing Registers

The microprocessor's register contents change continuously whenever the Apple ILe is running any sort of program, such as the Monitor. The Monitor lets you see what the register contents were when you invoked the Monitor or a program that you were debugging stopped at a break (BRK). The Monitor also lets you set 65C02 register values before you execute a program with the GO command.

When you call the Monitor, it stores the contents of the microprocessor's registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S (stack pointer), starting at location \$45 (decimal 69). When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

Pressing **CONTROL-E** and then **RETURN** invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

```
* CONTROL-E
```

```
A=0A X=FF Y=D8 P=B0 S=F8  
*:B0 02
```

```
* CONTROL-E
```

```
A=B0 X=02 Y=D8 P=B0 S=F8  
*
```

Monitor Cassette Tape Commands

The Apple IIe has two jacks for connecting an audio cassette tape recorder. With a recorder connected, you can use the Monitor commands described later in this section to save the contents of a range of memory onto a standard cassette and recall it for later use.

Saving Data on Tape

The Monitor's WRITE command saves the contents of up to 65,536 memory locations on cassette tape. To save a range of memory on tape, give the Monitor the starting and ending addresses of the range, followed by the letter W (for WRITE), like this:

```
{start} . {end} W
```

Don't press **RETURN** yet: first, put the tape recorder in record mode and let the tape run for a second, then press **RETURN**. The Monitor will write a ten-second tone onto the tape and then write the data. The tone acts as a leader: later, when the Monitor reads the tape, the leader enables the Monitor to get in step with the signal from the tape. When the Monitor is finished writing the range you specified, it will sound a bell (beep) and display a prompt. You should rewind the tape and label it with the memory range that's on the tape and what it's supposed to be.

Here's a small example you can save and use later to try out the READ command. Remember that you must start the cassette recorder in record mode before you press **RETURN** after typing the WRITE command.

```
*0:FF FF AD 30 C0 88 D0 04 C6 01 F0 08 CA
D0 F6 A6 00 4C 02 00 60
```

```
*0.14
```

```
0000- FF FF AD 30 C0 88 D0 04
0008- C6 01 F0 08 CA D0 F6 A6
0010- 00 4C 02 00 60
```

```
*0.14W
```

```
*
```

It takes about 35 seconds total to save the values of 4,096 memory locations preceded by the ten-second leader onto tape. This works out to an average data transfer rate of about 1,350 bits per second.

The WRITE command writes one extra value on the tape after it has written the values in the memory range. This extra value is the checksum, which is the eight-bit partial sum of all values in the range. When the Monitor reads the tape, it uses this value to determine if the data has been written and read correctly. (See the next section.)

Reading Data From Tape

Once you've saved a memory range onto tape with the Monitor's WRITE command, you can read that memory range back into the computer by using the Monitor's READ command. The data values you've stored on the tape need not be read back into the same memory range from whence they came; you can tell the Monitor to put those values into any memory range in the computer's memory, provided that it's the same size as the range you saved.

The format of the READ command is the same as that of the WRITE command, except that the command letter is R:

{start} . {end} R

Once again, after typing the command, don't press **RETURN**. Instead, start the tape recorder in play mode and wait a few seconds. Although the WRITE command puts a ten-second leader tone on the beginning of the tape, the READ command needs only three seconds of this leader to lock on to the signal from the tape. You should let a few seconds of tape go by before you press **RETURN** to allow the tape recorder's output to settle down to a steady tone.

This example has two parts. First, you set a range of memory to zero, verify the contents of memory, and then type the READ command, but don't press **RETURN**.

```
*0:000000000000000000000000
```

```
*0.14
```

```
0000- 00 00 00 00 00 00 00 00
```

```
0008- 00 00 00 00 00 00 00 00
```

```
0010- 00 00 00 00 00
```

```
*0.14R
```

Now start the cassette running in play mode, wait a few seconds, and press **RETURN**. After the Monitor sounds the bell (beep) and displays the prompt, examine the range of memory to see that the values from the tape were read correctly:

```
*0.14
```

```
0000- FF FF AD 30 C0 88 D0 04
```

```
0008- C6 01 F0 08 CA D0 F6 A6
```

```
0010- 00 4C 02 00 60
```

```
*
```

After the Monitor has read all the data values on the tape, it reads the checksum value. It computes the checksum on the data it read and compares it to the checksum from the tape. If the two checksums differ, the Monitor sends a beep to the speaker and displays **ERR**. This warns you that there was a problem reading the tape and that the values stored in memory aren't the values that were recorded on the tape. If the two checksums match, the Monitor will just send out a beep and display a prompt.

Miscellaneous Monitor Commands

These Monitor commands enable you to change the video display format from normal to inverse and back, and to assign input and output to accessories in expansion slots.

Inverse and Normal Display

You can control the setting of the inverse-normal mask location used by the COUT subroutine (described in Chapter 3) from the Monitor so that all of the Monitor's output will be in inverse format. The INVERSE command, I, sets the mask such that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command, N.

*O.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*I

*O.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*N

*O.F

0000- 0A 0B 0C 0D 0E 0F D0 04

0008- C6 01 F0 08 CA D0 F6 A6

*

Back to BASIC

Use the BASIC command, **CONTROL-B**, to leave the Monitor and enter the BASIC that was active when you entered the Monitor. Normally, this is Applesoft BASIC, unless you deliberately switched to Integer BASIC. Any program or variables that you had previously in BASIC will be lost. If you want to reenter BASIC with your previous program and variables intact, use the CONTINUE BASIC command, **CONTROL-C**.

If you are using DOS 3.3 or ProDOS, press **CONTROL-RESET** or type

3D0G

to return to the language you were using, with your program and variables intact.

That's a Number Not a Letter: If you use 3D0G, make sure that the third character you type is a zero, not a letter *O*. The letter *G* is the Monitor's GO command, described in the section "Machine-Language Programs" later in this chapter.

Redirecting Input and Output

The PRINTER command, activated by a **CONTROL-P**, diverts all output normally destined for the screen to an interface card in a specified expansion slot, from 1 to 7. There must be an interface card in the specified slot, or you will lose control of the computer and your program and variables may be lost. The format of the command is

{slot number} **CONTROL-P**

A PRINTER command to slot number 0 will switch the stream of output characters back to the Apple IIe's video display.

▲Warning

Don't give the PRINTER command with slot number 0 to deactivate the 80-column firmware, even though you used this command to activate it in slot 3. The command works, but it just disconnects the firmware, leaving some of the soft switches set for 80-column display.

In much the same way that the PRINTER command switches the output stream, the KEYBOARD command substitutes the interface card in a specified expansion slot for the Apple IIe's normal input device, the keyboard. The format for the KEYBOARD command is

{slot number} **CONTROL-K**

A slot number of 0 for the KEYBOARD command directs the Monitor to accept input from the Apple IIe's built-in keyboard.

The PRINTER and KEYBOARD commands are the exact equivalents of the BASIC commands PR# and IN#.

Hexadecimal Arithmetic

The Monitor will also perform one-byte hexadecimal addition and subtraction. Just type a line in one of these formats:

$$\begin{array}{l} | \text{value} | + | \text{value} | \\ | \text{value} | - | \text{value} | \end{array}$$

The Apple IIe performs the arithmetic and displays the result, as shown in these examples:

```
*20+13
=33
*4A-C
=3E
*FF+4
=03
*3-4
=FF
*
```

Special Tricks With the Monitor

This section describes some more complex ways of using the Monitor commands.

Multiple Commands

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces and the total number of characters in the line is less than 254. Adjacent single-letter commands such as L, S, I, and N need not be separated by spaces.

You can freely intermix all of the commands except the STORE (:) command. Since the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a STORE must be followed by a letter command before another address is encountered. You can use the NORMAL command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 N 300.302
```

```
0300- 00 00 00 00 00 00 00 00
```

```
0300- 18 69 01
```

```
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then grinds to a halt with a noisy beep and ignores the remainder of the input line.

Filling Memory

The MOVE command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range:

```
*300:11 22 33
```

```
*
```

Remember the number of values in the pattern: in this case, it is 3. Use the number to compute addresses for the MOVE command, like this:

|start+number| < |start| . |end-number| M

This MOVE command will first replicate the pattern at the locations immediately following the original pattern, then replicate that pattern following itself, and so on until it fills the entire range.

```
*303<300.32DM
```

```
*300.32F
```

```
0300- 11 22 33 11 22 33 11 22
```

```
0308- 33 11 22 33 11 22 33 11
```

```
0310- 22 33 11 22 33 11 22 33
```

```
0318- 11 22 33 11 22 33 11 22
```

```
0320- 33 11 22 33 11 22 33 11
```

```
0328- 22 33 11 22 33 11 22 33
```

```
*
```


You can do a similar trick with the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, you first fill the memory range from \$0300 to \$0320 with zeros and verify it, then change one location and verify again, to see the VERIFY command detect the discrepancy:

```
*300:0
*301<300.31FM
*301<300.31FV
*304:02
*301<300.31FV
0303-00 (02)
0304-02 (00)
*
```

Repeating Commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as N, and ending it with the sequence 34:n, where n is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, n=0. The value for n must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location \$0200. Each time the Monitor executes a command, it stores the value of the index at location \$34; when that command is finished, the Monitor reloads the index register with the value at location \$34. By making the last command change the value at location \$34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press **CONTROL-RESET**; that is how this example ends.

*N 300 302 34:0

0300- 11

0302- 33

0300- 11

0302- 33

0300- 11

0302- 33

0300- 11

0302- 33

0300- 11

0302- 33

0300- 11

0302- 33

030

Creating Your Own Commands

The `USER` command, `CONTROL-Y`, forces the Monitor to jump to memory location `$03F8`. You can put a `JMP` instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the `CONTROL-Y`. The program starts at location `$0300`; the command line that starts with `$03F8` stores a jump to `$0300` at location `$03F8`.

*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0 F5 4C 69 FF

*3F8:4C 00 03

* **CONTROL** **Y** THIS IS A TEST

THIS IS A TEST

Machine-Language Programs

The main reason to program in machine language is to get more speed. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

Boning Up on Machine Language: If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it and study at least one of the books on 6502 programming listed in the bibliography. With the books and Appendix A, you'll have the needed information to program the 65C02.

You can get a hexadecimal dump of your program, move it around in memory, or save it on tape and recall it using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

Running a Program

The Monitor command you use to start execution of your machine-language program is the GO command. When you type an address and the letter G, the Apple IIe starts executing machine language instructions starting at the specified location. If you just type the G, execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters A through Z: you store it starting at location \$0300, examine it to be sure you typed it correctly, then type 300G to start it running.

*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60

*300.30C

0300- A9 C1 20 ED FD 18 69 01

0308- C9 DB D0 F6 60

*300G

ABCDEFGHIJKLMNOPQRSTUVWXYZ

*

Disassembled Programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called **assemblers**.

Since programs that translate assembly language into machine language are called assemblers, a program like the Monitor's LIST command that translates machine language into assembly language is called a **disassembler**.

The Monitor's LIST command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or **mnemonic**, and a formatted hexadecimal operand. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format

[location] L

The word **mnemonic** comes from the same root as *memory* and refers to abbreviations that are easier to remember than the hexadecimal operation codes themselves: for example, for *clear carry* you write CLC instead of \$18.

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenfull (20 lines) of instructions, as shown in the following example:

*300L

```
0300-   A9 C1           LDA   #$C1
0302-   20 ED FD      JSR   $FDED
0306-   18            CLC
0306-   69 01          ADC   #$01
0308-   C9 DB          CMP   #$DB
030A-   D0 F6          BNE   $0302
030C-   60            RTS
030D-   00            BRK
030E-   00            BRK
030F-   00            BRK
0310-   00            BRK
0311-   00            BRK
0312-   00            BRK
0313-   00            BRK
0314-   00            BRK
0316-   00            BRK
0316-   00            BRK
0317-   00            BRK
0318-   00            BRK
0319-   00            BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has zeros in it; other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the program counter, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it will display another screenful of instructions, starting where the previous display left off.

The Mini-Assembler

Without an assembler, you have to write your machine language program, take the hexadecimal values for the opcodes and operands, and store them in memory using the commands covered in the previous sections. That is exactly what you did when you ran the previous examples.

The Monitor includes an assembler called the Mini-Assembler that lets you enter machine-language programs directly from the keyboard of your Apple. ASCII characters can be entered in Mini-Assembler programs, exactly as you enter them in the Monitor. Note that the Mini-Assembler doesn't accept labels; you must use actual values and addresses.

Starting the Mini-Assembler

To start the Mini-Assembler first invoke the Monitor by typing `CALL - 15 1` `RETURN`, and then from the Monitor, type `!` followed by `RETURN`. The Monitor prompt character then changes from `*` to `!`.

When you finish using the Mini-Assembler, press `RETURN` from a blank line to return to the Monitor.

Restrictions

The Mini-Assembler supports only the subset of 65C02 instructions that are found on the 6502.

Original Iie

Before you can use the Mini-Assembler on the original Apple Iie, you have to be running Integer BASIC. When you start up the computer using DOS or either BASIC, the Apple Iie loads the Integer BASIC interpreter from the file named INTBASIC into the bank-switched RAM. Here's how to start the Mini-Assembler on an original Apple Iie:

1. Start Integer BASIC from DOS 3.3 by typing `INT` `RETURN`.
2. After the Integer prompt character (`>`) and a cursor appear, enter the Monitor by typing `CALL - 15 1` `RETURN`.
3. Now start the Mini-Assembler by typing `F666G` `RETURN`.

Using the Mini-Assembler

The Mini-Assembler saves one address, that of the program counter. Before you start to type a program, you must set the program counter to point to the location where you want the Mini-Assembler to store your program. Do this by typing the address followed by a colon.

After the colon, type the mnemonic for the first instruction in your program, followed by a space and the operand of the instruction. Now press **RETURN**. The Mini-Assembler converts the line you typed into hexadecimal, stores it in memory beginning at the location of the program counter, and then disassembles it again and displays the disassembled line. It then displays a prompt on the next line.

Now the Mini-Assembler is ready to accept the second instruction in your program. To tell it that you want the next instruction to follow the first, don't type an address or a colon: just type a space and the next instruction's mnemonic and operand, then press **RETURN**. The Mini-Assembler assembles that line and waits for another.

Formats for operands are listed in Table 5-1.

```
!300:LDX #02
```

```
0300-    A2 02          LDX    #02
! LDA $0,X
```

```
0302-    B5 00          LDA    $00,X
! STA $10,X
```

```
0304     95 10          STA    $10,X
! DEX
```

```
0306-    CA            DEX
! STA $C030
```

```
0307-    8D 30 C0       STA    $C030
! BPL $302
```

```
030A-    10 F6          BPL    $0302
! BRK
```

```
030C-    00            BRK
!
```

If the line you type has an error in it, the Mini-Assembler beeps loudly and displays a caret (^) under or near the offending character in the input line. Most common errors are the result of typographical mistakes: misspelled mnemonics, missing parentheses, and so forth. The Mini-Assembler also rejects the input line if you forget the space before or after a mnemonic or

include an extraneous character in a hexadecimal value or address. If the destination address of a branch instruction is out of the range of the branch (more than 127 locations distant from the address of the instruction), the Mini-Assembler flags this as an error.

There are several different ways to leave the Mini-Assembler and reenter the Monitor. On an enhanced Apple IIe only, simply press **RETURN** at a blank line.

Original Iie | On an original Apple IIe, type the Monitor command \$FF69G.

On any Apple IIe, you can press **CONTROL**-**RESET**, which warm starts BASIC, then type

CALL -151

Your assembly-language program is now stored in memory. You can display it with the LIST command:

*3001

```
0300- A2 02      LDX    #$02
0302- B5 00      LDA    $00,X
0304- 95 10      STA    $10,X
0306- CA        DEX
0307- 8D 30 C0   STA    $C030
030A- 10 F6      BPL    $0302
030C- 00        BRK
030D- 00        BRK
030E- 00        BRK
030F- 00        BRK
0310- 00        BRK
0311- 00        BRK
0312- 00        BRK
0313- 00        BRK
0314- 00        BRK
0316- 00        BRK
0316- 00        BRK
0317- 00        BRK
0318- 00        BRK
0319- 00        BRK
```

*

Mini-Assembler Instruction Formats

See Appendix A for more information about 65C02 (and 6502) instructions.

The Apple Mini-Assembler recognizes 56 mnemonics and 13 addressing formats. These constitute the 6502 subset of the 65C02 instruction set. The mnemonics are standard, as used in the *Synertek Programming Manual* (Apple part number A2L0003), but the addressing formats are somewhat different. Table 5-1 shows the Apple standard address-mode formats for 6502 assembly language.

Table 5-1. Mini-Assembler Address Formats

Addressing Mode	Format
Accumulator	*
Implied	*
Immediate	#{value}
Absolute	\${address}
Zero page	\${address}
Indexed zero page	\${address},X \${address},Y
Indexed absolute	\${address},X \${address},Y
Relative	\${address}
Indexed indirect	(\${address},X)
Indirect indexed	(\${address}),Y
Absolute indirect	(\${address})

* These instructions have no operands.

An address consists of one or more hexadecimal digits. The Mini-Assembler interprets addresses the same way the Monitor does: if an address has fewer than four digits, the Mini-Assembler adds leading zeros; if the address has more than four digits, then it uses only the last four.

Dollar Signs: In this manual, dollar signs (\$) in addresses signify that the addresses are in hexadecimal notation. They are ignored by the Mini-Assembler and may be omitted when typing programs.

There is no syntactical distinction between the absolute and zero-page addressing modes. If you give an instruction to the Mini-Assembler that can be used in both absolute and zero-page mode, the Mini-Assembler assembles that instruction in absolute mode if the operand for that instruction is greater than \$FF, and it assembles it in zero-page mode if the operand is less than \$0100.

Instructions in accumulator mode and implied addressing mode need no operands.

Branch instructions, which use the relative addressing mode, require the target address of the branch. The Mini-Assembler calculates the relative distance to use in the instruction automatically. If the target address is more than 127 locations distant from the instruction, the Mini-Assembler sounds a bell (beep), displays a caret (^) under the target address, and does not assemble the line.

If you give the Mini-Assembler the mnemonic for an instruction and an operand, and the addressing mode of the operand cannot be used with the instruction you entered, the Mini-Assembler will not accept the line.

Summary of Monitor Commands

Here is a summary of the Monitor commands, showing the syntax for each one.

Examining Memory

{adrs}

Examines the value contained in one location.

{adrs1}..{adrs2}

Displays the values contained in all locations between {adrs1} and {adrs2}.

RETURN

Displays the values in up to eight locations following the last opened location.

Changing the Contents of Memory

`{adr}|:{val}|{val}|...`

Stores the values in consecutive memory locations starting at `{adr}|`.

`:{val}|{val}|...`

Stores values in memory starting at the next changeable location.

Moving and Comparing

`{dest}|<|{start}|.|end|`M

Copies the values in the range `{start}|.|end|` into the range beginning at `{dest}|`.

`{dest}|<|{start}|.|end|`V

Compares the values in the range `{start}|.|end|` to those in the range beginning at `{dest}|`.

The Examine Command

`CONTROL` `E`

Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.

The Search Command

`{val}|<|{start}|.|end|`S

Displays the address of the first occurrence of `{val}|` in the specified range beginning at `{start}|`.

Cassette Tape Commands

`{start}|.|end|`W

Writes the values in the memory range `{start}|.|end|` onto tape, preceded by a ten-second leader.

`{start}|.|end|`R

Reads values from tape, storing them in memory beginning at `{start}|` and stopping at `{end}|`. Prints **ERR** if an error occurs.

Miscellaneous Monitor Commands

I	Sets inverse display mode.
N	Sets normal display mode.
CONTROL B	Enters the language currently active (usually Applesoft).
CONTROL C	Returns to the language currently active (usually Applesoft).
val + val	Adds the two values and prints the hexadecimal result.
val - val	Subtracts the second value from the first and prints the result.
slot CONTROL P	Diverts output to the device whose interface card is in slot number slot . If slot =0, accepts input from the keyboard.
CONTROL Y	Jumps to the machine-language subroutine at location \$3F8.

Running and Listing Programs

ads G	Transfers control to the machine language program beginning at ads .
ads L	Disassembles and displays 20 instructions, starting at ads . Subsequent LIST commands display 20 more instructions.

The Mini-Assembler

Original Iie

The Mini-Assembler is available on an original Apple Iie only when Integer BASIC is active. See the earlier section “The Mini-Assembler.”

F666G	Invokes the Mini-Assembler on the original Apple Iie.
!	Invokes the Mini-Assembler on the enhanced Apple Iie.
\$(command	Executes a Monitor command from the Mini-Assembler on the original Apple Iie.
\$FF69G	Leaves the Mini-Assembler on the original Apple Iie.
RETURN	Leaves the Mini-Assembler on the enhanced Apple Iie.



The seven expansion slots on the Apple IIe's main circuit board are used for installing circuit cards containing the hardware and firmware needed to interface peripheral devices to the Apple IIe. These slots are not simple I/O ports; peripheral cards can access the Apple IIe's data, address, and control lines via these slots. The expansion slots are numbered from 1 to 7, and certain signals, described below, are used to select a specific slot.

II Plus, II

The Apple II and Apple II Plus have an eighth expansion slot: slot number 0. On those models, slot 0 is normally used for a language card or a ROM card; the functions of the Apple II Language Card are built into the main circuit board of the Apple IIe.

Interrupt support on the enhanced Apple IIe requires that special attention be paid to cards designed to be in slot 3. A description of what you need to watch for is given at the end of this chapter.

Original IIe

The interrupt support built into the enhanced Apple IIe is an enhanced and expanded version of the interrupt support in the original Apple IIe.

Peripheral-Card Memory Spaces

Because the Apple IIe's microprocessor does all of its I/O through memory locations, portions of the Apple IIe's memory space have been allocated for the exclusive use of the cards in the expansion slots. In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or PROM) on the peripheral cards themselves.

The memory spaces allocated for the peripheral cards are described below. Those memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware like this are called intelligent peripherals. They make it possible for you to add peripheral hardware to your Apple IIe without having to change your programs, provided that your programs follow normal practice for data input and output.

Peripheral-Card I/O Space

Each expansion slot has the exclusive use of sixteen memory locations for data input and output in the memory space beginning at location \$C090. Slot 1 uses locations \$C090 through \$C09F, slot 2 uses locations \$C0A0 through \$C0AF, and so on through location \$C0FF, as shown in Table 6-1.

Signals for which the active state is low are marked with a prime (').

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple IIe addresses one of the sixteen I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called DEVICE SELECT', switches to the active (low) state. This signal can be used to enable logic on the peripheral card that uses the four low-order address lines to determine which of its sixteen I/O locations is being accessed.

Table 6-1. Peripheral-Card I/O Memory Locations Enabled by DEVICE SELECT'

Slot	Locations	Slot	Locations
1	\$C090-\$C09F	5	\$C0D0-\$C0DF
2	\$C0A0-\$C0AF	6	\$C0E0-\$C0EF
3	\$C0B0-\$C0BF	7	\$C0F0-\$C0FF
4	\$C0C0-\$C0CF		

Peripheral-Card ROM Space

One 256-byte page of memory space is allocated to each accessory card. This space is normally used for read-only memory (ROM or PROM) on the card with driver programs that control the operation of the peripheral device connected to the card.

The page of memory allocated to each expansion slot begins at location \$Cn00, where n is the slot number, as shown in Table 6-2 and Figure 6-3. Whenever the Apple IIe addresses one of the 256 ROM memory locations allocated to a particular slot, the signal on pin 1 of that slot, called I/O SELECT', switches to the active (low) state. This signal enables the ROM or PROM devices on the card, and the eight low-order address lines determine which of the 256 memory locations is being accessed.

Table 6-2. Peripheral-Card ROM Memory Locations Enabled by I/O SELECT'

Slot	Locations	Slot	Location
1	\$C100-\$C1FF	5	\$C500-\$C5FF
2	\$C200-\$C2FF	6	\$C600-\$C6FF
3	\$C300-\$C3FF	7	\$C700-\$C7FF
4	\$C400-\$C4FF		

Expansion ROM Space

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K-byte memory space from \$C800 to \$CFFF for larger programs in ROM or PROM. This memory space is called expansion ROM space. (See the memory map in Figure 6-3). Besides being larger, the expansion ROM memory space is always at the same locations regardless of which slot is occupied by the card, making programs that occupy this memory space easier to write.

This memory space is available to any peripheral card that needs it. More than one peripheral card can have expansion ROM on it, but only one of them can be active at a time.

Each peripheral card that uses expansion ROM must have a circuit on it to enable the ROM. The circuit does this by a two-stage process: first, it sets a flip-flop when the I/O SELECT' signal, pin 1 on the slot, becomes active (low); second, it enables the expansion ROM devices when the I/O STROBE' signal, pin 20 on the slot, becomes active (low). Figure 6-1 shows a typical ROM-enable circuit.

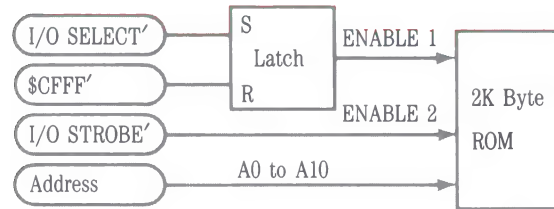
The I/O SELECT' signal on a particular slot becomes active whenever the Apple IIe's microprocessor addresses a location in the 256-byte ROM address space allocated to that slot. The I/O STROBE' signal on all of the expansion slots becomes active (low) when the microprocessor addresses a location in the expansion-ROM memory space, \$C800-\$CFFF. The I/O STROBE' signal is used to enable the expansion-ROM devices on a peripheral card. (See Figure 6-1.)

Important!

If there is an 80-column text card installed in the auxiliary slot, some of the functions normally associated with slot 3 are performed by the 80-column text card and the built-in 80-column firmware. With the 80-column text card installed, the I/O STROBE' signal is not available on slot 3, so firmware in expansion ROM on a card in slot 3 will not run.

See the section "I/O Programming Suggestions" later in this chapter.

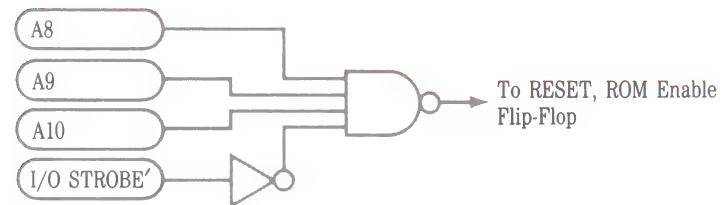
Figure 6-1. Expansion ROM Enable Circuit



A program on a peripheral card can get exclusive use of the expansion ROM memory space by referring to location \$CFFF in its initialization phase. This location is special: all peripheral cards that use expansion ROM must recognize a reference to \$CFFF as a signal to reset their ROM-enable flip-flops and disable their expansion ROMs. Of course, doing so also disables the expansion ROM on the card that is about to use it, but the next instruction in the initialization code sets the flip-flop in the expansion-ROM enable circuit on the card.

A card that needs to use the expansion ROM space must first insert its slot address (\$Cn) in \$07F8 before it refers to \$CFFF. This allows interrupting devices to reenable the card's expansion ROM after interrupt handling is finished. Once its slot address has been inserted in \$07F8, the peripheral card has exclusive use of the expansion memory space and its program can jump directly into the expansion ROM.

Figure 6-2. ROM Disable Address Decoding



As described earlier, the expansion-ROM disable circuit resets the enable flip-flop whenever the 65C02 addresses location \$CFFF. To do this, the peripheral card must detect the presence of \$CFFF on the address bus. You can use the I/O STROBE' signal for part of the address decoding, since it is active for addresses from \$C800 through \$CFFF. If you can afford to sacrifice some ROM space, you can simplify the address decoding even further and save circuitry on the card. For example, if you give up the last 256 bytes of expansion ROM space, your disable circuit only needs to detect addresses of the form \$CFxx, and you can use the minimal disable-decoding circuitry shown in Figure 6-2.

Important!

Applesoft addresses two locations in the \$CFxx space, thereby resetting the enable flip-flop. If your peripheral device is going to be used with Applesoft programs, you must either use the full address decoding or else enable the expansion ROM each time it is needed.

Peripheral-Card RAM Space

There are 56 bytes of main memory allocated to the peripheral cards, eight bytes per card, as shown in Table 6-3. These 56 locations are actually in the RAM memory reserved for the text and low-resolution graphics displays, but these particular locations are not displayed on the screen and their contents are not changed by the built-in output routine COUT1. Programs in ROM on peripheral cards use these locations for temporary data storage.

Table 6-3. Peripheral-Card RAM Memory Locations

Base Address	Slot Number						
	1	2	3*	4	5	6	7
\$0478	\$0479	\$047A	\$047B*	\$047C	\$047D	\$047E	\$047F
\$04F8	\$04F9	\$04FA	\$04FB*	\$04FC	\$04FD	\$04FE	\$04FF
\$0578	\$0579	\$057A	\$057B*	\$057C	\$057D	\$057E	\$057F
\$05F8	\$05F9	\$05FA	\$05FB*	\$05FC	\$05FD	\$05FE	\$05FF
\$0678	\$0679	\$067A	\$067B*	\$067C	\$067D	\$067E	\$067F
\$06F8	\$06F9	\$06FA	\$06FB*	\$06FC	\$06FD	\$06FE	\$06FF
\$0778	\$0779	\$077A	\$077B*	\$077C	\$077D	\$077E	\$077F
\$07F8	\$07F9	\$07FA	\$07FB*	\$07FC	\$07FD	\$07FE	\$07FF

* If there is a card in the auxiliary slot, it takes over these locations.

A program on a peripheral card can use the eight base addresses shown in the table to access the eight RAM locations allocated for its use, as shown in the next section, “I/O Programming Suggestions.”

▲Warning

The Apple IIe firmware sets the value of \$04FB to \$FF on a reset, even if there is no 80-column card installed.

I/O Programming Suggestions

A program in ROM on a peripheral card should work no matter which slot the card occupies. If the program includes a jump to an absolute location in one of the 256-byte memory spaces, then the card will work only when it is plugged into the slot that uses that memory space. If you are writing the program for a peripheral card that will be used by many people, you should avoid placing such a restriction on the use of the card.

Important!

To function properly no matter which slot a peripheral card is installed in, the program in the card's 256-byte memory space must not make any absolute references to itself. Instead of using jump instructions, you should force conditions on branch instructions, which use relative addressing.

The first thing a peripheral-card used as an I/O device must do when called is to save the contents of the Apple IIe's microprocessor's registers. (Peripheral cards not being used as I/O devices do not need to save the registers.) The device should save the register's contents on the stack, and restore them just before returning control to the calling program. If there is RAM on the peripheral card, the information may be stored there.

Most single-character I/O is done via the microprocessor's accumulator. A character being output through your subroutine will be in the accumulator with its high bit set when your subroutine is called. Likewise, if your subroutine is performing character input, it must leave the character in the accumulator with its high bit set when it returns to the calling program.

Finding the Slot Number With ROM Switched In

The memory addresses used by a program on a peripheral card differ depending on which expansion slot the card is installed in. Before it can refer to any of those addresses, the program must somehow determine the correct slot number. One way to do this is to execute a JSR (jump to subroutine) to a location with an RTS (return from subroutine) instruction in it, and then derive the slot number from the return address saved on the stack, as shown in the following example.

```
PHP          ; save status
SEI          ; inhibit interrupts
JSR KNOWNRTS ; -> a known RTS instruction...
              ; ...that you set up
TSX          ; get high byte of the...
LDA $0100,X  ; ...return address from stack
AND #$0F     ; low-order digit is slot no.
PLP          ; restore status
```

The slot number can now be used in addressing the memory allocated to the peripheral card, as shown in the next section.

I/O Addressing

Once your peripheral-card program has the slot number, the card can use the number to address the I/O locations allocated to the slot. Table 6-4 shows how these locations are related to sixteen base addresses starting with \$C080. Notice that the difference between the base address and the desired I/O location has the form \$n0, where n is the slot number. Starting with the slot number in the accumulator, the following example computes this difference by four left shifts, then loads it into an index register and uses the base address to specify one of sixteen I/O locations.

```
ASL          ; get n into...
ASL          ;
ASL          ;
ASL          ; ...high-order nybble...
TAX          ; ... of index register.
LDA $C080,X  ; load from first I/O location
```

See the section “Setting Bank Switches” in Chapter 4 for more information.

Selecting Your Target: You must make sure that you get an appropriate value into the index register when you address I/O locations this way. For example, starting with 1 in the accumulator, the instructions in the above example perform an LDA from location \$C090, the first I/O location allocated to slot 1. If the value in the accumulator had been 0, the LDA would have accessed location \$C080, thereby setting the soft switch that selects the second bank of RAM at location \$D000 and enables it for reading.

Table 6-4. Peripheral-Card I/O Base Addresses

Base Address	Connector Number						
	1	2	3	4	5	6	7
\$C080	\$C090	\$C0A0	\$C0B0	\$C0C0	\$C0D0	\$C0E0	\$C0F0
\$C081	\$C091	\$C0A1	\$C0B1	\$C0C1	\$C0D1	\$C0E1	\$C0F1
\$C082	\$C092	\$C0A2	\$C0B2	\$C0C2	\$C0D2	\$C0E2	\$C0F2
\$C083	\$C093	\$C0A3	\$C0B3	\$C0C3	\$C0D3	\$C0E3	\$C0F3
\$C084	\$C094	\$C0A4	\$C0B4	\$C0C4	\$C0D4	\$C0E4	\$C0F4
\$C085	\$C095	\$C0A5	\$C0B5	\$C0C5	\$C0D5	\$C0E5	\$C0F5
\$C086	\$C096	\$C0A6	\$C0B6	\$C0C6	\$C0D6	\$C0E6	\$C0F6
\$C087	\$C097	\$C0A7	\$C0B7	\$C0C7	\$C0D7	\$C0E7	\$C0F7
\$C088	\$C098	\$C0A8	\$C0B8	\$C0C8	\$C0D8	\$C0E8	\$C0F8
\$C089	\$C099	\$C0A9	\$C0B9	\$C0C9	\$C0D9	\$C0E9	\$C0F9
\$C08A	\$C09A	\$C0AA	\$C0BA	\$C0CA	\$C0DA	\$C0EA	\$C0FA
\$C08B	\$C09B	\$C0AB	\$C0BB	\$C0CB	\$C0DB	\$C0EB	\$C0FB
\$C08C	\$C09C	\$C0AC	\$C0BC	\$C0CC	\$C0DC	\$C0EC	\$C0FC
\$C08D	\$C09D	\$C0AD	\$C0BD	\$C0CD	\$C0DD	\$C0ED	\$C0FD
\$C08E	\$C09E	\$C0AE	\$C0BE	\$C0CE	\$C0DE	\$C0EE	\$C0FE
\$C08F	\$C09F	\$C0AF	\$C0BF	\$C0CF	\$C0DF	\$C0EF	\$C0FF

RAM Addressing

A program on a peripheral card can use the eight base addresses shown in Table 6-3 to access the eight RAM locations allocated for its use. The program does this by putting its slot number into the Y index register and using indexed addressing mode with the base addresses. The base addresses can be defined as constants because they are the same no matter which slot the peripheral card occupies.

If you start with the correct slot number in the accumulator (by using the example shown earlier), then the following example uses all eight RAM locations allocated to the slot.

```
TAY
LDA    $0478,Y
STA    $04F8,Y
LDA    $0578,Y
STA    $05F8,Y
LDA    $0678,Y
STA    $06F8,Y
LDA    $0778,Y
STA    $07F8,Y
```

▲Warning

You must be very careful when you have your peripheral-card program store data at the base-address locations themselves since they are temporary storage locations; the RAM at those locations is used by the disk operating system. Always store the first byte of the ROM location of the expansion slot that is currently active (\$Cn) in location \$7F8, and the first byte of the ROM location of the slot holding the controller card for the startup disk drive in location \$5F8.

Changing the Standard I/O Links

There are two pairs of locations in the Apple IIe that are used for controlling character input and output. They are called the I/O links. In a Apple IIe running without a disk operating system, the I/O links normally contain the starting addresses of the standard input and output routines—KEYIN and COUT1 if the 80-column firmware is not active, BASICIN and BASICOUT if the 80-column is active. If a disk operating system is running, one or both of the links will hold the addresses of the operating system input and output routines.

See "The Standard I/O Links" in Chapter 3.

The link at locations \$36 and \$37 (decimal 54 and 55) is called CSW, for *character output switch*. Individually, location \$36 is called CSWL (CSW Low) and location \$37 is called CSWH (CSW High). CSW holds the starting address of the subroutine the Apple IIe is currently using for single-character output. This address is normally \$FDF0, the address of routine COUT1, or \$C307, the address of BASICOUT.

COUT1 and BASICOUT are described in Chapter 3.

When you issue a PR#n from BASIC or an n **CONTROL**-P from the Monitor, the Apple IIe changes this link address to the first address in the ROM memory space allocated to slot number n. That address has the form \$Cn00. Subsequent calls for character output are thus transferred to the program on the peripheral card. That program can use the instruction sequences given above to find its slot number and use the I/O and RAM locations allocated to it. When it is finished, the program can execute an RTS (return from subroutine) instruction to return control to the calling program, or jump to the output routine COUT1 at location \$FDF0 to display the output character (which must be in the accumulator) on the screen, then let COUT1 return to the calling program.

A similar link at locations \$38 and \$39 (decimal 56 and 57) is called KSW, for *keyboard input switch*. Individually, location \$38 is called KSWL (for KSW low) and location \$39 is called KSWH (KSW high). KSW holds the starting address of the routine currently being used for single-character input. This address is normally \$FD1B, the starting address of KEYIN, or \$C305, the address of BASICIN.

KEYIN and BASICIN are described in Chapter 3.

When you issue an IN#n command from BASIC or an n **CONTROL**-[K] from the Monitor, the Apple IIe changes this link address to \$Cn00, the beginning of the ROM memory space that is allocated to slot number n. Subsequent calls for character input are thus transferred to the program on the accessory card. That program can use the instruction sequences given above to find its slot number and use the I/O and RAM locations allocated to it. The program should put the input character, with its high bit set, into the accumulator and execute an RTS instruction to return control to the program that requested input.

When a disk operating system (ProDOS or DOS 3.3) is running, one or both of the standard I/O links hold addresses of the operating system's input and output routines. The operating system has internal locations that hold the addresses of the character input and output routines that are currently active.

Important!

See the *ProDOS Technical Reference Manual* for more about using link addresses.

Refer to the section on input and output link registers in the *DOS Programmer's Manual* and the *ProDOS Technical Reference Manual* for further details.

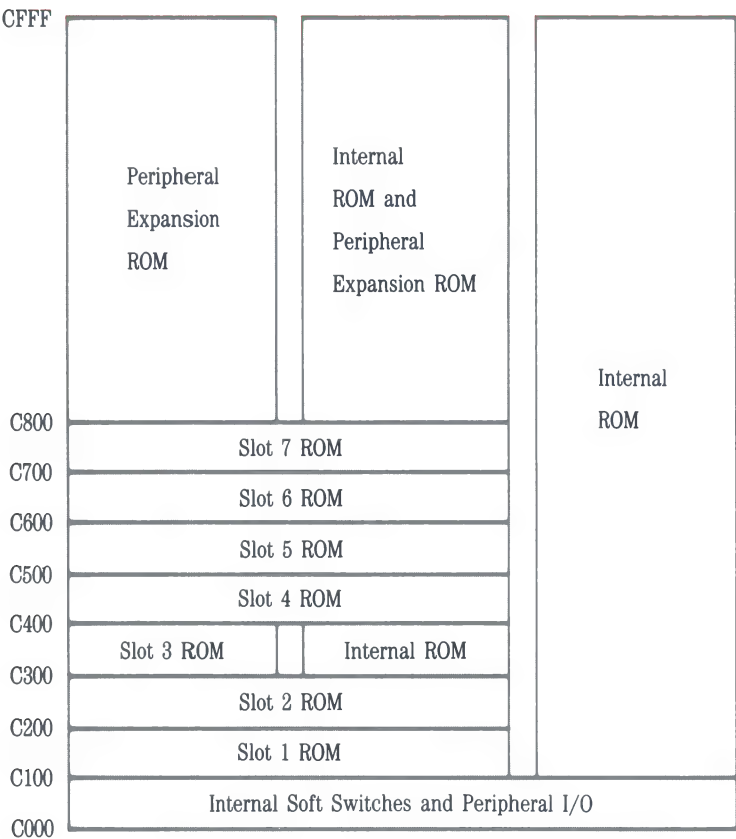
If a program that is running with ProDOS or DOS 3.3 changes the standard link addresses, either directly or via IN# and PR# commands, the operating system is disconnected.

To avoid disconnecting the operating system each time a BASIC program initiates I/O to a slot, it should use either an IN# or a PR# command from inside a PRINT statement that starts with a Control-D character. For assembly-language programs, there is a DOS 3.3 subroutine call to use when changing the link addresses. After changing CSW or KSW, the program calls this subroutine at location \$03EA (decimal 1002). The subroutine transfers the link address to a location inside the operating system and then restores the operating system address in the standard link location.

Other Uses of I/O Memory Space

The portion of memory space from location \$C000 through \$CFFE' (decimal 49152 through 53247) is normally allocated to I/O and program memory on the peripheral cards, but there are two other functions that also use this memory space: the built-in self-test firmware and the 80-column display firmware. The soft switches that control the allocation of this memory space are described in the next section.

Figure 6-3. I/O Memory Map



Switching I/O Memory

The built-in firmware uses two soft switches to control the allocation of the I/O memory space from \$C000 to \$CFFF. The locations of these soft switches, SLOTCXROM and SLOTC3ROM, are given in Table 6-5.

Note: Like the display switches described in Chapter 2, these soft switches share their locations with the keyboard data and strobe functions. The switches are activated only by writing, and the states can be determined only by reading, as indicated in Table 6-5.

Table 6-5. I/O Memory Switches

Name	Function	Hex	Location		Notes
			Hex	Decimal	
SLOT3ROM	Slot ROM at \$C300	\$C00B	49163	-16373	Write
	Internal ROM at \$C300	\$C00A	49162	-16374	Write
	Read SLOT3ROM switch	\$C017	49175	-16361	Read
SLOT4ROM	Slot ROM at \$C400	\$C006	49159	-16377	Write
	Internal ROM at \$C400	\$C007	49158	-16378	Write
	Read SLOT4ROM switch	\$C015	49173	-16363	Read

When SLOT3ROM is on, the 256-byte ROM area at \$C300 is available to a peripheral card in slot 3, which is the slot normally used for a terminal interface. If a card is installed in the auxiliary slot when you turn on the power or reset the Apple IIe, the SLOT3ROM switch is turned off. Turning SLOT3ROM off disables peripheral-card ROM in slot 3 and enables the built-in 80-column firmware, as shown in Figure 6-3. The 80-column firmware is assigned to slot-3 address space because slot 3 is normally used with a terminal interface, so the built-in firmware will work with programs that use slot 3 this way.

The bus and I/O signals are always available to a peripheral card in slot 3, even when the 80-column hardware and firmware are operating. Thus it is always possible to use this slot for any I/O peripheral that does *not* have built-in firmware.

When SLOT4ROM is active (high), the I/O memory space from \$C100 to \$C7FF is allocated to the expansion slots, as described previously. Setting SLOT4ROM inactive (low) disables the peripheral-card ROM and selects built-in ROM in all of the I/O memory space except the part from \$C000 to \$C0FF (used for soft switches and data I/O), as shown in Figure 6-3. In addition to the 80-column firmware at \$C300 and \$C800, the built-in ROM includes firmware that performs the self-test of the Apple IIe's hardware.

Note: Setting SLOT4ROM low enables built-in ROM in all of the I/O memory space (except the soft-switch area), including the \$C300 space, which contains the 80-column firmware.

Developing Cards for Slot 3

Original ILe

In the original Apple ILe firmware, the internal slot 3 firmware was always switched in if there was an 80-column card (either 1K or 64K) in the auxiliary slot. This means that peripheral cards with their own ROM were effectively switched out of slot 3 when the system was turned on.

With the enhanced Apple ILe Monitor ROM, the rules are different. A peripheral card in slot 3 is now switched in when the system is started up or when **RESET** is pressed *if* the card's ROM has the following ID bytes:

\$C305 = \$38

\$C307 = \$18

The enhanced Apple ILe firmware requires that interrupt code be present in the \$C3 page (either external or internal). A peripheral card in slot 3 must have the following code to support interrupts. After this segment, the code continues execution in the internal ROM at \$C400.

```
$C3F4:  IRQDONE      STA $C081      ;Read ROM, write RAM
                                JMP $FC7A      ;Jump to $F8 ROM

                                IRQ
                                BIT $C015      ;slot or internal ROM
                                STA $C007      ;force in internal ROM
```

When programming for cards in slot 3:

- ☐ You must support the AUXMOVE and XFER routines at \$C312 and \$C314.
- ☐ Don't use unpublished entry points into the internal \$Cn00 firmware, because there is no guarantee that they will stay the same.
- ☐ If your peripheral card is a character I/O device, you must follow the Pascal 1.1 firm ware protocol, described in the next section.

For more information about the \$C300 firmware, see the Monitor ROM listing in Appendix I of this manual. Especially note the portion from \$C300 through \$C420.

Pascal 1.1 Firmware Protocol

The Pascal 1.1 firmware protocol was originally developed to be used with Apple Pascal 1.1 programs. The protocol is followed by all succeeding versions of Apple II Pascal, and can be used by programmers using other languages as well.

The Pascal 1.1 firmware protocol provides Apple IIe programmers with

- a standard way to uniquely identify new peripheral cards
- a standard way to address the firmware routines in peripheral cards.

Device Identification

The Pascal 1.1 firmware protocol uses four bytes near the beginning of the peripheral card's firmware to identify the peripheral card.

Address	Value
---------	-------

\$Cs05	\$38 (like the old Apple II Serial Interface Card)
\$Cs07	\$18 (like the old Apple II Serial Interface Card)
\$Cs0B	\$01 (the generic signature of new cards)
\$Cs0C	\$ci (the device signature)

The first hexadecimal digit, c, of the device signature byte identifies the device class and the second hexadecimal digit, i, of the device signature byte is a unique identifier for the card, used by some manufacturers for their cards. Table 6-6 shows the device class assignments.

Table 6-6. Peripheral-Card Device-Class Assignment

Digit	Device Class
\$0	Reserved
\$1	Printer
\$2	Joystick or other X-Y input device
\$3	Serial or parallel I/O card
\$4	Modem
\$5	Sound or speech device
\$6	Clock
\$7	Mass storage device
\$8	80-column card
\$9	Network or bus interface
\$A	Special purpose (none of the above)
\$B-F	Reserved for future expansion

For example, the Apple II Super Serial Card has a device signature of \$31: the 3 signifies that it is a serial or parallel I/O card, and the 1 is the low-order digit supplied by Apple Technical Support.

Although version 1.1 of Pascal ignores the device signature, applications programs can use them to identify specific devices.

I/O Routine Entry Points

Indirect calls to the firmware in a peripheral card are done through a branch table in the card's firmware. The branch table of I/O routine entry points is located near the beginning of the Cs00 address space (s being the slot number where the peripheral card is installed).

The branch table locations that Pascal 1.1 firmware protocol uses are as follows:

Address	Contains
\$Cs0D	Initialization routine offset (required)
\$Cs0E	Read routine offset (required)
\$Cs0F	Write routine offset (required)
\$Cs10	Status routine offset (required)
\$Cs11	\$00 if optional offsets follow; non-zero if not
\$Cs12	Control routine offset (optional)
\$Cs13	Interrupt handling routine offset (optional)

Notice that \$Cs11 contains \$00 only if the control and interrupt handling routines are supported by the firmware. (For example, the SSC does not support these two routines, and so location \$Cs11 contains a non-zero firmware instruction.) Apple II Pascal 1.0 and 1.1 do not support control and interrupt requests, but such requests are implemented in Pascal 1.2 and later versions and in ProDOS.

Table 6-7 gives the entry point addresses and the contents of the 65C02 registers on entry to and on exit from Pascal 1.1 I/O routines.

Table 6-7. I/O Routine Offsets and Registers Under Pascal 1.1 Protocol

Addr.	Offset for	X Register	Y Register	A Register
\$Cs0D	Initialization			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	(unchanged)
\$Cs0E	Read			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	Character read
\$Cs0F	Write			
	On entry	\$Cs	\$s0	Char. to write
	On exit	Error code	(unchanged)	(unchanged)
\$Cs10	Status			
	On entry	\$Cs	\$s0	Request (0 or 1)
	On exit	Error code	(changed)	(unchanged)

Interrupts on the Enhanced Apple IIe

The original Apple IIe offered little firmware support for interrupts. The enhanced Apple IIe's firmware provides improved interrupt support, very much like the Apple IIc's interrupt support. Neither machine disables interrupts for extended periods.

Interrupts work on enhanced Apple IIe systems with an installed 80-column text card (either 1K or 64K) or a peripheral card with interrupt-handling ROM in slot 3. Interrupts are easiest to use with ProDOS and Pascal 1.2 because they have interrupt support built in. DOS 3.3 has no built-in interrupt support.

The new interrupt handler operates like the Apple IIc interrupt handler, using the same memory locations and operating protocols. The main purpose of the interrupt handler is to support interrupts in *any* memory configuration. This is done by saving the machine's state at the time of the interrupt, placing the Apple in a standard memory configuration before calling your program's interrupt handler, then restoring the original state when your program's interrupt handler is finished.

For more about interrupt support in ProDOS, see the *ProDOS Technical Reference Manual*.

For information about interrupt handling with Apple Pascal 1.2, see the *Device and Interrupt Support Tools Manual* which is part of the Apple II Device Support Tools package (A2W0014).

What Is an Interrupt?

An interrupt is a hardware signal that tells the computer to stop what it is currently doing and devote its attention to a more important task. Print spooling and mouse handling are examples of interrupt use, things that don't take up all the time available to the system, but that should be taken care of promptly to be most useful.

For example, the Apple IIe mouse can send an interrupt to the computer every time it moves. If you handle that interrupt promptly, the mouse pointer's movement on the screen will be smooth instead of jerky and uneven.

Interrupt priority is handled by a daisy-chain arrangement using two pins, INT IN and INT OUT, on each peripheral-card slot. As described in Chapter 7, each peripheral card breaks the chain when it makes an interrupt request. On peripheral cards that don't use interrupts, these pins should be connected together.

The daisy chain gives priority to the peripheral card in slot 7: if this card opens the connection between INT IN and INT OUT, or if there is no card in this slot, interrupt requests from cards in slots 1 through 6 can't get through. Similarly, slot 6 controls interrupt requests (IRQ) from slots 1 through 5, and so on down the line.

When the IRQ' line on the Apple IIe's microprocessor is activated (pulled low), the microprocessor transfers control through the vector in locations \$FFFE-\$FFFF. This vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an external IRQ or a BRK instruction and transfers control to the appropriate routine via the vectors stored in memory page 3. The BRK vector is in locations \$03F0-\$03F1 and ProDOS uses the IRQ vector in locations \$03FE-\$03FF. (See Table 4-11.) The Monitor normally stores the address of its reset routine in the IRQ vector; you should substitute the address of your program's interrupt-handling routine.

Apple Pascal doesn't use the BRK vector at \$03F0-\$03F1, but it does use the IRQ vector at \$03FE-\$03FF.

Interrupts on Apple II Series Computers

The interrupt handler built in to the enhanced Apple IIe's firmware saves the contents of the accumulator on the stack. (The original Apple IIe saves the contents of the accumulator at location \$45.) DOS 3.3, as well as the Monitor, rely on the integrity of location \$45, so this change lets both DOS 3.3 and the Monitor continue to work with active interrupts on the enhanced Apple IIe.

Original IIe

Since the built-in interrupt handler on the original Apple IIe uses location \$45 to save the contents of the accumulator, the operating system fails when an interrupt occurs under DOS 3.3 on the original Apple IIe.

If you want to write programs that use interrupts while running on the original Apple IIe, Apple II Plus, or Apple II, you must use either ProDOS or Apple II Pascal 1.2 (or later versions). Both these operating systems give you full interrupt support, even though these versions of the Apple II don't include interrupt support in their firmware. (Versions of Pascal before 1.2 do not work with interrupts enabled on an original Apple IIe.)

Some other manufacturer's hardware, such as co-processor cards, don't work properly in an interrupting environment. If you are trying to develop an application and encounter this problem, check with the manufacturer of the card to see if a later version of the hardware or its software will operate properly with interrupts active. You may not be able to use interrupts if an interrupt-tolerant version isn't available.

Interrupts are effective only if they are enabled most of the time. Interrupts that occur while interrupts are disabled will not be serviced.

Pascal, DOS 3.3, and ProDOS turn off interrupts while performing disk operations because of the critical timing of disk read and write operations. Some peripheral cards used in the Apple IIe disable interrupts while reading and writing.

Original IIe

Although the enhanced Apple IIe firmware never disables interrupts during screen handling, the original Apple IIe periodically turns interrupts off while doing 80-column screen operations. The effect is most noticeable while the screen is scrolling.

Important!

Don't use PR#6 to restart your Apple IIe while running ProDOS with interrupts enabled since PR#6 doesn't disable interrupts. If you try it, ProDOS will fail as it starts up since its interrupt handlers aren't yet set up. If you have to restart, use **CONTROL-RESET**, or make sure that your program disables interrupts before it ends.

Rules of the Interrupt Handler

Unlike the Apple IIc, the enhanced Apple IIe's interrupt handling firmware is not always switched in. Here are the reasons why this is so and the implications that necessarily follow.

There is *no* part of memory in the Apple IIe that is always switched in. Thus, there is no location for an interrupt handler that works for all memory configurations. However, the \$C3 page of firmware is present on all systems that have 80-column text cards in their auxiliary slots, so it was selected as the starting location of the built-in interrupt handling routine.

There are two factors that determine if the \$C3 firmware is switched in and therefore whether or not interrupts will be usable:

- ☐ Is there an 80-column text card in the auxiliary slot?
- ☐ If not, is there a peripheral card in slot 3 with built-in ROM with bytes \$C305 = \$38 and \$C307 = \$18?

The Apple IIe's memory is switched according to the following rules at both powerup and reset:

- ☐ If there is a ROM card in slot 3, but no text card in the auxiliary slot, the firmware on the ROM card is switched in. This is necessary for Pascal to work.
- ☐ If there is a text card in the auxiliary slot, but no ROM card in slot 3, the internal \$C3 firmware is switched in.
- ☐ If there is both a text card in the auxiliary slot and a ROM card in slot 3, the firmware on the ROM card is switched in.

Important!

These rules mean that systems without 80-column text cards in the auxiliary slot do not have their internal \$C3 firmware switched in. Such systems cannot handle interrupts or breaks (the software equivalent of interrupts). An application program must swap in the \$C3 firmware both on initialization and after reset to make interrupts function properly on such a machine configuration. (ProDOS versions 1.1 and later do this for you during startup.)

See the section "Developing Cards for Slot 3" earlier in this chapter.

Another implication of the decision to have interrupt code in the \$C3 page affects the shared \$C800 space in the Apple IIe. When the \$C3 page is referenced, the IIe hardware automatically switches in its own \$C800 space. When the interrupt handler finishes, it restores the \$C800 space to the original owner using MSLOT (\$07F8). This means that it is very important for a peripheral card to place its slot address in MSLOT to support interrupts while code is being executed in its \$C800 space.

Interrupt Handling on the 65C02 and 6502

There are three possible conditions that will allow interrupts on the 65C02 and 6502:

- The IRQ line on the microprocessor is pulled low after a CLI instruction has been used (interrupts are not masked). This is the standard technique that devices use when they need immediate attention.
- The microprocessor executes a break instruction (BRK = opcode \$00).
- A non-maskable interrupt (NMI) occurs. The microprocessor services this interrupt whether or not the CLI instruction has been used. An NMI is completely independent of the interrupts discussed in this manual.

The microprocessor saves the current program counter and status byte on the stack when an interrupt occurs and then jumps to the routine whose address is stored in \$FFFE and \$FFFF. The sequence of operations performed by the microprocessor is as follows:

1. It finishes executing the current instruction if an IRQ is encountered. (If a BRK instruction is encountered, the current instruction is already finished.)
2. It pushes the high byte of the program counter onto the stack.
3. It pushes the low byte of the program counter onto the stack.
4. It pushes the processor status byte onto the stack.
5. It executes a JMP (\$FFFE) instruction.

The Interrupt Vector at \$FFFE

Three separate regions of memory contain address \$FFFE in an Apple IIe with an Extended 80-Column Text Card: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at \$FFFE in the ROM points to the built-in interrupt handling routine. You must copy the ROM's interrupt vector to the other banks yourself if you plan to use interrupts with the bank-switched memory switched in.

The Built-in Interrupt Handler

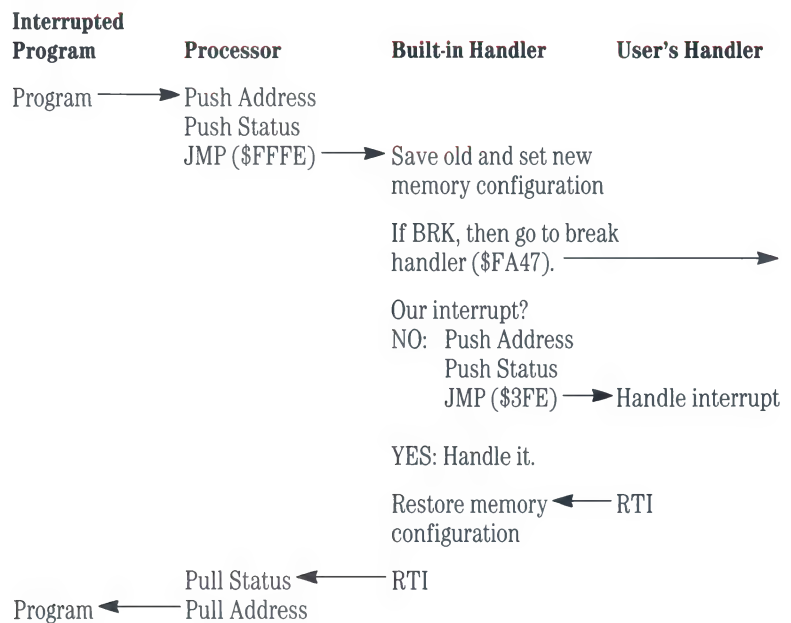
The enhanced Apple IIe's built-in interrupt handler records the computer's current memory configuration, then sets the computer's memory configuration to a standard state so that your program's interrupt handler always begins running in the same memory configuration.

Next the built-in interrupt handler checks to see if the interrupt was caused by a break instruction, and handles it as just described under "Interrupt Handling on the 65C02 and 6502." If it was not a break, it passes control to the interrupt handling routine whose address is stored at \$3FE and \$3FF of main memory. Normally, that would be the operating system's interrupt handler, unless you have installed one of your own.

After your program's interrupt handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does another RTI to return to where it was when the interrupt occurred. Figure 6-4 illustrates this entire process. Each of these steps is explained later in this chapter.

Interrupt handler installation is described in the *ProDOS Technical Reference Manual* and the *Device and Interrupt Support Tools Manual*, which is part of the Apple IIe Device Support Tools package (A2W0014).

Figure 6-4. Interrupt-Handling Sequence



Saving the Apple IIe's Memory Configuration

The built-in interrupt handler saves the Apple IIe's memory configuration and then sets it to a known state according to these rules:

- Text Page 1 is switched in (PAGE2 off) so that main screen holes are accessible if 80STORE and PAGE2 are on.
- Main memory is switched in for reading (RAMRD off).
- Main memory is switched in for writing (RAMWRT off).
- \$D000-\$FFFF ROM is switched in for reading (RDLCRAM off).
- Main stack and zero page are switched in (ALTZP off).
- The auxiliary stack pointer is preserved, and the main stack pointer is restored. (See the next section, "Managing Main and Auxiliary Stacks.")

Important!

Because main memory is switched in, all memory addresses used later in this chapter are in main memory unless otherwise specified.

Managing Main and Auxiliary Stacks

Apple has adopted a convention that allows the Apple IIe to be run with two separate stack pointers since the Apple IIe with an Extended 80-Column Text Card has two stack pages. Two bytes in the auxiliary stack page are used as storage for inactive stack pointers: \$0100 for the main stack pointer when the auxiliary stack is active, and \$0101 for the auxiliary stack pointer when the main stack is active.

When a program using interrupts switches in the auxiliary stack for the first time, it must place the value of the main stack pointer at \$0100 (in the auxiliary stack) and initialize the auxiliary stack pointer to \$FF (the top of the stack). When it subsequently switches from one stack to the other, it must save the current stack pointer before loading the pointer for the other stack.

The current stack pointer is stored at \$0101, and the main stack pointer is retrieved from \$0100 when an interrupt occurs while the auxiliary stack is switched in. *Then* the main stack is switched in for use. The stack pointer is restored to its original value after the interrupt has been handled.

Important!

The built-in XFER routine does not support this procedure. If you are using XFER to swap stacks, you must use code like the following to set up the stack pointers and stack.

```
*
* This example transfers control from a code segment running
* using the main stack to one running using the aux stack.
*

1  XFERALT  PHP                ;preserve interrupt status in A
2          PLA
3          SEI                ;disable interrupts
4          TSX                ;save main stack pointer at $100
5          STA SETALTZP       ;and swap zero pages
6          STX $100
7          LDX $101          ;now restore aux stack pointer
8          TXS
9          PHA                ;and interrupt status
10         PLP

11         LDA #DESTL        ;set destination address
12         STA $3ED
13         LDA #DESTH
14         STA $3EE
15         SEC/CLC            ;set direction of transfer
16         BIT RTS            ;V=1 for alt zero page (RTS=$60)
17         JMP XFER          ;do transfer
```

To transfer control the other direction, change the following lines

```
5          STX $101
6          LDX $100
7          STA SETSTDZP

16         CLV                ;V=0 for main zp
```


The User's Interrupt Handler at \$03FE

If your program has an interrupt handler, it must place the entry address of that handler at \$03FE. After it sets the machine to a standard state, the IIe's internal interrupt handler transfers control to the routine whose address is in the vector at \$03FE.

It is very important for a peripheral card to place its slot address in MSL0T to support interrupts whenever it is executing code in its \$C800 space. Whenever the \$C3 page is referenced, the IIe automatically switches in its own \$C800 ROM space. When the interrupt handler finishes, it restores the \$C800 space to the original owner using MSL0T (\$07F8).

▲Warning

Be careful to install interrupt handlers according to the rules of the operating system that you are using. Placing the address of your program's interrupt handler at \$03FE disconnects the operating system's interrupt handler.

The \$03FE interrupt handler must do these things:

1. Verify that the interrupt came from the expected source.
2. Handle the interrupt as desired.
3. Clear the appropriate interrupt soft switch.
4. Return with an RTI.

Here are some things to remember if you are dealing with programs that must run in an interrupt environment:

- There is no guaranteed maximum response time for interrupts because the system may be doing a disk operation that lasts for several seconds.
- Once the built-in interrupt handler is called, it takes *at least* 150 to 200 microseconds for it to call your interrupt handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.
- If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst state. (The worst state is one that requires the most work to set up for: 80STORE and PAGE2 on; auxiliary memory switched in for reading and writing; bank-switched memory page 2 in the auxiliary bank switched in for reading and writing; and internal \$Cn00 ROM switched in).
- Interrupt overhead will be greater if your interrupt handler is installed through an operating system's interrupt dispatcher. The length of delay depends on the operating system, and on whether the operating system dispatches the interrupt to other routines before calling yours.

Handling Break Instructions

The 65C02 treats a break instruction (BRK, opcode \$00) just like a hardware interrupt. After the interrupt handler sets the memory configuration, it checks to see if the interrupt was caused by a break (bit 4 of the status byte is set), and if it was, jumps to a break handling routine. This routine saves the state of the computer at the time of the break as shown in Table 6-8.

Table 6-8. BRK Handler Information

Information	Location
Program counter (low byte)	\$3A
Program counter (high byte)	\$3B
Encoded memory state	\$44
Accumulator	\$45
X register	\$46
Y register	\$47
Status register	\$48

Finally the break routine jumps to the routine whose address is stored at \$3F0 and \$3F1.

The encoded memory state in location \$44 is interpreted as shown in Table 6-9.

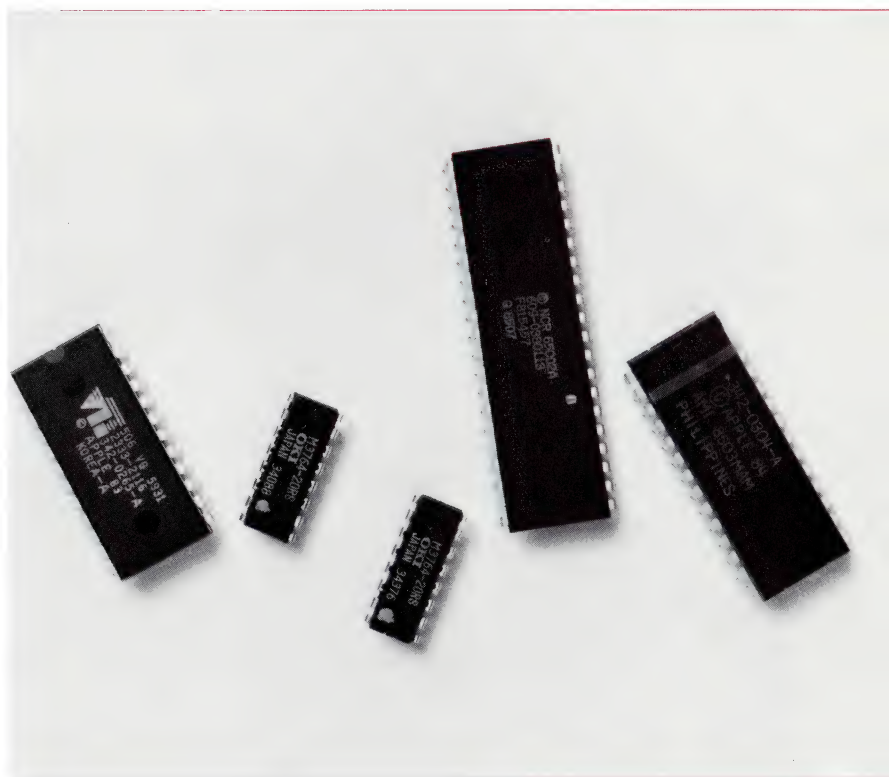
Table 6-9. Memory Configuration Information

Bit 7 = 1	if auxiliary zero page and auxiliary stack are switched in
Bit 6 = 1	if 80STORE and PAGE2 both on
Bit 5 = 1	if auxiliary RAM switched in for reading
Bit 4 = 1	if auxiliary RAM switched in for writing
Bit 3 = 1	if bank-switched RAM being read
Bit 2 = 1	if bank-switched \$D000 Page 1 switched in and RAMREAD set
Bit 1 = 1	if bank-switched \$D000 Page 2 switched in and RAMREAD set
Bit 0 = 1	if internal Cs ROM was switched in (Ile only)

Interrupt Differences: Apple IIe Versus Apple IIc

If you are writing software for both the Apple IIe and the Apple IIc, you should know that there are several important differences between the interrupts on the enhanced Apple IIe and those on the Apple IIc. They are

- In the IIc ROM, \$FFFE points to \$C803; in the IIe ROM, to \$C3FA. To ensure that the proper interrupt vectors are placed into the Language Card RAM space, always copy them to the RAM from the ROM. (When you initialize built-in devices on the IIc, these vectors are automatically updated).
- There is no shared \$C800 ROM in the IIc. Peripheral cards share this space in the IIe. Thus it is crucial that the slot address of the peripheral card using the \$C800 space is stored in MSL0T (\$07F8). When the interrupt handler goes to the internal \$C3 space, the IIe hardware switches in its own \$C800 space. When the interrupt handler finishes, it restores the \$C800 space to the slot whose address is in MSL0T.
- The IIc \$C800 space is always switched in. The enhanced IIe's interrupt handler preserves the state of the \$C800-space switch and then switches in the slot I/O space. This means that when restoring the state of the system using the value placed in location \$44, break handling routines must restore one more value on the Apple IIe than on the Apple IIc.



Most of this manual describes functions—what the Apple IIe does. This chapter, on the other hand, describes objects: the pieces of hardware the Apple IIe uses to carry out its functions. If you are designing a piece of peripheral hardware to attach to the Apple IIe, or if you just want to know more about how the Apple IIe is built, you should study this chapter.

Environmental Specifications

The Apple IIe is quite sturdy when used in the way it was intended. Table 7-1 defines the conditions under which the Apple IIe is designed to function properly.

Table 7-1. Summary of Environmental Specifications

Operating Temperature:	0° to 45° C (30° to 115° F)
Relative Humidity:	5% to 85%
Line Voltage:	107 to 132 VAC

You should treat the Apple IIe with the same kind of care as any other electrical appliance. You should protect it from physical violence, such as hammer blows or defenestration. You should protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids, especially those with dissolved contaminants, such as coffee and cola drinks.

In normal operation, enough air flows through the slots in the case to keep the insides from getting too hot, although some of the parts inside the Apple IIe normally get rather warm to the touch. If you manage to overheat your Apple IIe, by blocking the ventilation slots in the top and bottom for example, the first symptom will be erratic operation. The memory devices in the Apple IIe are sensitive to heat: when they get too hot, they occasionally change a bit of data. The exact result depends on what kind of program you are running and on just which bit of memory is affected.

The Power Supply

The power supply in the Apple IIe operates on normal household AC power and provides enough low-voltage electrical power for the built-in electronics plus a full complement of peripheral cards, including disk controller cards and communications interfaces. The basic specifications of the power supply are listed in Table 7-2.

The Apple IIe's power cord should be plugged into a three-wire 110- to 120-volt outlet. You must connect the Apple IIe to a grounded outlet or to a good earth ground. Also, the line voltage must be in the range given in Table 7-2. If you try to operate the Apple IIe from a power source with more than 140 volts, you will damage the power supply.

Table 7-2. Power Supply Specifications

Line voltage:	107V to 132V AC
Maximum power consumption:	60W continuous 80W intermittent*
Supply voltages:	+5V \pm 3% +11.8V \pm 6% -5.2V \pm 10% -12V \pm 10%
Maximum supply currents:	+5V: 2.5A +12V: 1.5A continuous, 2.5A intermittent* -5V: 250mA -12V: 250mA
Maximum case temperature:	55° C (130° F)

* Intermittent operation: The Apple IIe can safely operate for up to twenty minutes at the higher load if followed by at least ten minutes at normal load.

The Apple IIe uses a custom-designed switching-type power supply. It is small and lightweight, and it generates less heat than other types of power supplies do.

The Apple IIe's power supply works by converting the AC line voltage to DC and using this DC voltage to power a variable-frequency oscillator. The oscillator drives a small transformer with many separate windings to produce the different voltages required. A circuit compares the voltage of the +5-volt supply with a reference voltage and feeds an error signal back to the oscillator circuit. The oscillator circuit uses the error signal to control the frequency of its oscillation and keep the output voltages in their normal ranges.

The power supply includes circuitry to protect itself and the other electronic parts of the Apple IIe by turning off all four supply voltages whenever it detects one of the following malfunctions:

- ☐ any supply voltage short-circuited to ground
- ☐ the power-supply cable disconnected
- ☐ any supply voltage outside the normal range

Any time one of these malfunctions occurs, the protection circuit stops the oscillator, and all the output voltages drop to zero. After about half a second, the oscillator starts up again. If the malfunction is still occurring, the protection circuit stops the oscillator again. The power supply will continue to start and stop this way until the malfunction is corrected or the power is turned off.

▲Warning

If you think the power supply is broken, do not attempt to repair it yourself. The power supply is in a sealed enclosure because some of its circuits are connected directly to the power line. Special equipment is needed to repair the power supply safely, so see your authorized Apple dealer for service.

The Power Connector

The cable from the power supply is connected to the main circuit board by a six-pin connector with a strain-relief catch. The connector pins are identified in Table 7-3 and Figure 7-13d.

Table 7-3. Power Connector Signal Specifications

Pin Number	Name	Description
1,2	Ground	Common electrical ground
3	+5V	+5V from power supply
4	+12V	+12V from power supply
5	-12V	-12V from power supply
6	-5V	-5V from power supply

The 65C02 Microprocessor

The enhanced Apple IIe uses a 65C02 microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIe runs at a clock rate of 1.023 MHz and performs up to 500,000 eight-bit operations per second. You should not use the clock rate as a criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1MHz clock is equivalent to other types of microprocessors with clock rates up to 2.5MHz.

The 65C02 has a sixteen-bit address bus, giving it an address space of 64K (2 to the sixteenth power or 65536) bytes. The Apple IIe uses special techniques to address a total of more than 64K: see the sections “Bank-Switched Memory” and “Auxiliary Memory and Firmware” in Chapter 4 and the section “Switching I/O Memory” in Chapter 6.

See Appendix A for a description of the 65C02's instruction set and electrical characteristics.

Table 7-4. 65C02 Microprocessor Specifications

Type:	65C02
Register Complement:	8-bit Accumulator (A) 8-bit Index Registers (X,Y) 8-bit Stack Pointer (S) 8-bit Processor Status (P) 16-bit Program Counter (PC)
Data Bus:	Eight bits wide
Address Bus:	Sixteen bits wide
Address Range:	65,536 (64K)
Interrupts:	IRQ (maskable) NMI (non-maskable) BRK (programmed)
Operating Voltage:	+5V (\pm 5%)
Power Dissipation:	5 mW (at 1 MHz)

65C02 Timing

The operation of the Apple IIe is controlled by a set of synchronous timing signals, sometimes called clock signals. In electronics, the word *clock* is used to identify signals that control the timing of circuit operations. The Apple IIe doesn't contain the kind of clock you tell time by, although its internal timing is accurate enough that a program running on the Apple IIe can simulate such a clock.

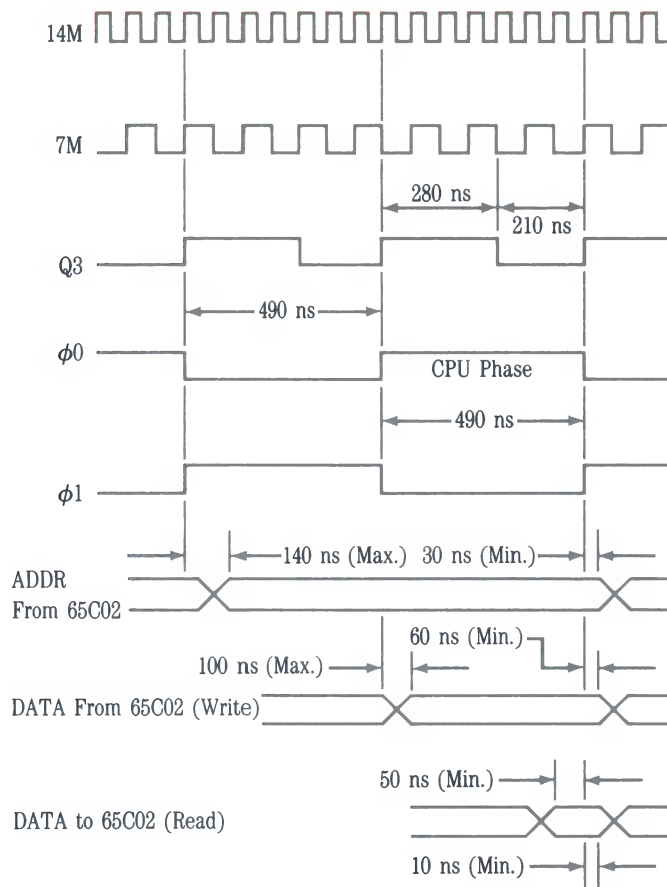
The frequency of the oscillator that generates the master timing signal is 14.31818 MHz. Circuitry in the Apple IIe uses this clock signal, called 14M, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the operation of the 65C02 (and 6502 on the original version of the Apple IIe) are described in this section. Other timing signals are described in this chapter in the sections "RAM Addressing," "Video Display Modes," and "The Expansion Slots."

The main 65C02 timing signals are listed in Table 7-5, and their relationships are diagrammed in Figure 7-1. The 65C02 clock signals are $\phi 1$ and $\phi 0$, complementary signals at a frequency of 1.02273 MHz. The Apple IIe signal named $\phi 0$ is equivalent to the signal called $\phi 2$ in the hardware manual. (It isn't identical: it's a few nanoseconds early.)

Table 7-5. 65C02 Timing Signal Descriptions

Signal Name	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
$\phi 0$	Phase 0 of 65C02 clock, 1.0227 MHz; complement of $\phi 1$
$\phi 1$	Phase 1 of 65C02 clock, 1.0227 MHz; complement of $\phi 0$

Figure 7-1. 65C02 Timing Signals



The operations of the 65C02 are related to the clock signals in a simple way: address during $\phi 1$, data during $\phi 0$. The 65C02 puts an address on the address bus during $\phi 1$. This address is valid not later than 140 nanoseconds after $\phi 1$ goes high and remains valid through all of $\phi 0$. The 65C02 reads or writes data during $\phi 0$. If the 65C02 is writing, the read/write signal is low during $\phi 0$ and the 65C02 puts data on the data bus. The data is valid not later than 75 nanoseconds after $\phi 0$ goes high. If the 65C02 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of $\phi 0$.

The Custom Integrated Circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIe is in three custom integrated circuits called the Memory Management Unit (MMU), the Input/Output Unit (IOU), and the Programmed Array Logic device (PAL). The soft switches used for controlling the various I/O and addressing modes of the Apple IIe are addressable flags inside the MMU and the IOU. The functions of these two devices are not as independent as their names suggest; working together, they generate all of the addressing signals. For example, the MMU generates the address signals for the CPU, while the IOU generates similar address signals for the video display.

The Memory Management Unit

The circuitry inside the MMU implements these soft switches, which are described in the indicated chapters in this manual:

- ☐ Page 2 display (PAGE2): Chapter 2
- ☐ High resolution mode (HIRES): Chapter 2
- ☐ Store to 80-column card (80STORE): Chapter 2
- ☐ Select bank 2: Chapter 4
- ☐ Enable bank-switched RAM: Chapter 4
- ☐ Read auxiliary memory (RAMRD): Chapter 4
- ☐ Write auxiliary memory (RAMWRT): Chapter 4
- ☐ Auxiliary stack and zero page (ALTZP): Chapter 4
- ☐ Slot ROM for connector #3 (SLOT3ROM): Chapter 6
- ☐ Slot ROM in I/O space (SLOTXROM): Chapter 6

The 64K dynamic RAMs used in the Apple IIe use a multiplexed address, as described later in this chapter in the section “Dynamic-RAM Timing.” The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU. The pinouts and signal descriptions of the MMU are shown in Figure 7-2 and Table 7-6.

Figure 7-2. The MMU Pinouts

GND	1	40	A1
A0	2	39	A2
ϕ 0	3	38	A3
Q3	4	37	A4
PRAS'	5	36	A5
RA0	6	35	A6
RA1	7	34	A7
RA2	8	33	A8
RA3	9	32	A9
RA4	10	31	A10
RA5	11	30	A11
RA6	12	29	A12
RA7	13	28	A13
R/W'	14	27	A14
INH'	15	26	A15
DMA'	16	25	+5V
EN80'	17	24	Cxxx
KBD'	18	23	RAMEN'
ROMEN2'	19	22	R/W' 245
ROMEN1'	20	21	MD7

Table 7-6. The MMU Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	ϕ 0	Clock phase 0 input
4	Q3	Timing signal input
5	PRAS'	Memory row-address strobe
6-13	RA0-RA7	Multiplexed address output
14	R/W'	65C02 read-write control signal
15	INH'	Inhibits main memory (tied to +5 V)
16	DMA'	Controls data bus for DMA transfers
17	EN80'	Enables auxiliary RAM
18	KBD'	Enables keyboard data bits 0-6
19	ROMEN2'	Enables ROM (tied to ROMEN1')
20	ROMEN1'	Enables ROM (tied to ROMEN2')
21	MD7	State of MMU flags on data bus bit 7
22	RW'245	Controls 74LS245 data-bus buffer
23	RAMEN'	Enables main RAM
24	Cxxx	Enables peripheral-card memory
25	+5V	Power
26-40	A15-A1	65C02 address input

The Input/Output Unit

The circuitry inside the Input/Output Unit (IOU) implements the following soft switches, all described in Chapter 2 in this manual:

- ☐ Page 2 display (PAGE2)
- ☐ High resolution mode (HIRES)
- ☐ Text mode (TEXT)
- ☐ Mixed mode (MIXED)
- ☐ 80-column display (80COL)
- ☐ Text display mode select (ALTCHAR)
- ☐ Any-key-down
- ☐ Annunciators
- ☐ Vertical blanking (VBL)

The 64K dynamic RAMs used in the Apple IIe require a multiplexed address, as described later in this chapter in the section “Dynamic-RAM Timing.” The IOU generates this multiplexed address for the data transfers required for display and memory refresh during clock phase 1. The way this address is generated is described later in this chapter in the section “Display Address Mapping.” The pinouts and signal descriptions for the IOU are shown in Figure 7-3 and Table 7-7.

Figure 7-3. The IOU Pinouts

GND	1	40	H0
GR	2	39	SYNC'
SEGA	3	38	WNDW'
SEGB	4	37	CLRGAT'
VC	5	36	RA10'
80VID'	6	35	RA9'
CASSO	7	34	VID6
SPKR	8	33	VID7
MD7	9	32	KSTRB
AN0	10	31	AKD
AN1	11	30	C0xx
AN2	12	29	A6
AN3	13	28	+5V
R/W'	14	27	Q3
RESET'	15	26	ϕ 0
(n.c.)	16	25	PRAS'
RA0	17	24	RA7
RA1	18	23	RA6
RA2	19	22	RA5
RA3	20	21	RA4

Table 7-7. The IOU Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects high-resolution when low, low-resolution when high
5	VC	Display vertical counter bit: in text mode, SEGA, SEGB and VC determine which of the eight rows of a character's dot pattern to display; in low-resolution, selects upper or lower block defined by a byte.
6	80VID'	80-column video enable
7	CASSO	Cassette output signal
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)3
10-13	AN0-AN3	Annunciator outputs
14	R/W'	65C02 read-write control signal
15	RESET'	Power on and reset output
16		Nothing is connected to this pin.
17-24	RA0-RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS'	Row-address strobe (phase 0)
26	ϕ 0	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	C0xx	I/O address enable
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33,34	VIDD7,VIDD6	Video display data bits
35,36	RA9',RA10'	Video display control bits
37	CLRGAT'	Color-burst gate (enable)
38	WNDW'	Display blanking signal
39	SYNC'	Display synchronization signal
40	H0	Display horizontal timing signal (low bit of character counter)

The PAL Device

A Programmed Array Logic device, type PAL 16R8, generates several timing and control signals in the Apple IIe. These signals are listed in Table 7-8. The PAL pinouts are given in Figure 7-4.

Figure 7-4. The PAL Pinouts

14M	1	20	+5V
7M	2	19	PRAS'
3.58M	3	18	(n.c.)
H0	4	17	PCAS'
VID7	5	16	Q3
SEGB	6	15	$\phi 0$
GR	7	14	$\phi 1$
RAMEN'	8	13	VID7M
80VID'	9	12	LDPS'
GND	10	11	ENTMG

Table 7-8. The PAL Signal Descriptions

Pin Number	Name	Description
1	14M	14.31818 MHz master timing signal
2	7M	7.15909 MHz timing signal
3	3.58M	3.579545 MHz timing signal
4	H0	Horizontal video timing signal
5	VID7	Video data bit 7
6	SEGB	Video timing signal
7	GR	Video display graphics-mode enable
8	RAMEN'	RAM enable (CAS enable)
9	80VID'	Enable 80-column display mode
10	GND	Power and signal common
11	ENTMG	Enable master timing
12	LDPS'	Video shift-register load enable
13	VID7M	Video dot clock, 7 or 14 MHz
14	$\phi 1$	Phase 1 system clock
15	$\phi 0$	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS'	RAM column-address strobe
18	N.C.	(This pin is not used.)
19	PRAS'	RAM row-address strobe
20	+5V	Power

Memory Addressing

The Apple IIe's microprocessor can address 65,536 locations. The Apple IIe uses this entire address space, and then some: some areas in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIe and the way they are addressed. Input and output also use portions of the memory address space; refer to the section "Peripheral-Card Memory Spaces" in Chapter 6 for information.

Figure 7-5. The 2364 ROM Pinouts

+5V	1	28	+5V
A12	2	27	+5V
A7	3	26	+5V
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	ROMENx'
A2	8	21	A10
A1	9	20	CE'
A0	10	19	MD7
MD0	11	18	MD6
MD1	12	17	MD5
MD2	13	16	MD4
GND	14	15	MD3

Figure 7-6. The 2316 ROM Pinouts

A7	1	24	+5V
A6	2	23	A8
A5	3	22	A9
A4	4	21	+5V
A3	5	20	KBD'
A2	6	19	GND
A1	7	18	ENKBD'
A0	8	17	(n.c.)
MD0	9	16	MD6
MD1	10	15	MD5
MD2	11	14	MD4
GND	12	13	MD3

Figure 7-7. The 2333 ROM Pinouts

VID4	1	24	+5V
VID3	2	23	VID5
VID2	3	22	RA9
VID1	4	21	GR
VID0	5	20	WNDW'
VC	6	19	RA10
SEGB	7	18	ENVID'
SEGA	8	17	D7
D0	9	16	D6
D1	10	15	D5
D2	11	14	D4
GND	12	13	D3

ROM Addressing

In the Apple IIe, the following programs are permanently stored in two type 2364 8K by 7-bit ROMs (read-only memory):

- ❑ Applesoft editor and interpreter
- ❑ System Monitor
- ❑ 80-column display firmware
- ❑ self-test routines

These two ROMs are enabled by two signals called ROMEN1 and ROMEN2. The ROM enabled by ROMEN1, sometimes called the Diagnostics ROM, occupies the memory address space from \$C100 to \$DFFF. The address space from \$C300 to \$C3FF and from \$C800 to \$CFFF contains the 80-column display firmware. Those address spaces are normally assigned to ROM on a peripheral card in slot 3; for a discussion of the way the 80-column firmware overrides the peripheral card, see the section "Other Uses of I/O Memory Space" in Chapter 6. The pinouts of the 2364 ROMs are given in Figure 7-5.

Two other portions of the Diagnostics ROM, addressed from \$C100 to \$C2FF and from \$C400 to \$C7FF, contain the built-in self-test routines. These address spaces are normally assigned to the peripheral cards; when the self-test programs are running, the peripheral cards are disabled.

The remainder of the Diagnostics ROM, addressed from \$D000 to \$DFFF, contains part of the Applesoft BASIC interpreter.

The ROM enabled by ROMEN2, sometimes called the Monitor ROM, occupies the memory address space from \$E000 to \$FFFF. This ROM contains the rest of the Applesoft interpreter, in the address space from \$E000 to \$EFFF, and the Monitor subroutines, from \$F000 to \$FFFF.

The other ROMs in the Apple IIe are a type 2316 ROM used for the keyboard character decoder and a type 2333 ROM used for character sets for the video display. This 2333 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry. The 2316's pinout is given in Figure 7-6, and the 2333's pinout is given in Figure 7-7.

Figure 7-8. The 64K RAM Pinouts

+5V	1	16	GND
MDx	2	15	CAS'
R/W'	3	14	MDx
RAS'	4	13	RA1
RA7	5	12	RA4
RA5	6	11	RA3
RA6	7	10	RA2
+5V	8	9	RA0

RAM Addressing

The RAM (programmable) memory in the Apple IIe is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIe, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIe take advantage of the two-phase system clock described earlier in this chapter in the section “65C02 Timing” to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during $\phi 0$, and the display circuits read data only during $\phi 1$.

Dynamic-RAM Refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIe reads the data in the active display page and sends it to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIe refreshes the display sixty times per second.

The dynamic RAM devices used in the Apple IIe also need a kind of refresh, because the data is stored in the form of electric charges which diminish with time and must be replenished every so often. The Apple IIe is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called multiplexing. Since only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs.

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described later in this chapter in the section “The Video Counters,” the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated eight times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every two milliseconds. (See Figure 7-9.) This more than satisfies the refresh requirements of the dynamic RAMs.

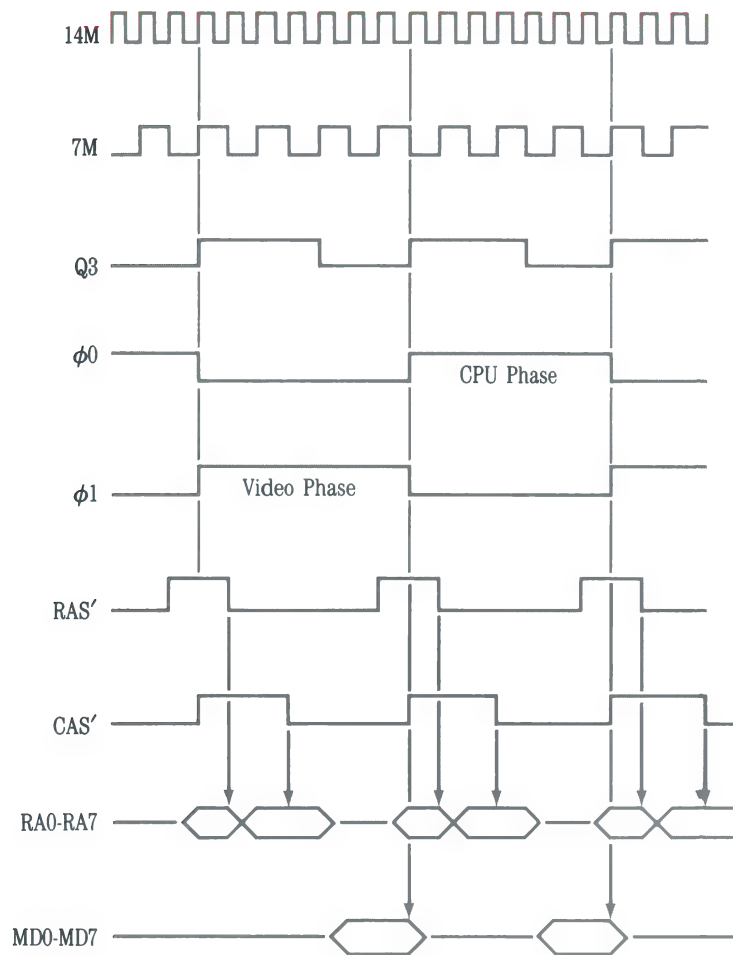
Table 7-9. RAM Address Multiplexing

Mux'd Address	Row Address	Column Address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

Dynamic-RAM Timing

The Apple IIe's microprocessor clock runs at a moderate speed, about 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2 MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU is strobed by the falling edge of $\phi 0$, and display data is strobed by the falling edge of $\phi 1$, as shown in Figure 7-9.

Figure 7-9. RAM Timing Signals



The RAM timing looks complicated because the RAM address is multiplexed, as described in the previous section. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines labelled RA0-RA7. Along with the other timing signals, the PAL device generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).

Table 7-10. RAM Timing Signal Descriptions

Signal Name	Description
$\phi 0$	Clock phase 0 (CPU phase)
$\phi 1$	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/column-address strobe
RA0-RA7	Multiplexed address bus
MD0-MD7	Internal data bus

The Video Display

The Apple IIe produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that is being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

Video Standards: Apple IIe's manufactured for sale in the U.S. generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIe's manufactured for sale in European countries generate video that is compatible with the standard used there, which is called P.A.L. (for phase alternating lines). This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named WNDW' is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the blanking intervals, the display is blank and the WNDW' signal is high. The synchronization signals, called *sync* for short, are produced by making the signal named SYNC' low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

The Video Counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE'. The input to the horizontal counter is the 1 MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count normally from 0 to 63, then start over at 0. Whenever this happens, HPE' forces another count with H0 through H5 held at zero, thus extending the total count to 65.

The IOU uses the forty horizontal count values from 25 through 64 in generating the low-order part of the display data address, as described later in this chapter in the section "Display Address Mapping." The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from zero. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 261 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIe's video display is not interlaced.)

Smooth Animation: Animation displays sometimes have an erratic flicker caused by changing the display data at the same time it is being displayed. You can avoid this on the Apple IIe by reading the vertical-blanking signal (VBL) at location \$C019 and changing display data while VBL is low only (data value less than 128).

Display Memory Addressing

As described in Chapter 2 in the section “Addressing Display Pages Directly,” data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data is stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal \$400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing it directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in the section “Video Display Pages” in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary. They will not (alas!) eliminate that necessity.

The address transformation that folds three rows of forty display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are then described in the section “Video Display Modes.”

Display Address Mapping

Consider the simplest display on the Apple IIe, the 40-column text mode. To address forty columns requires six bits, and to address twenty-four rows requires another five bits, for a total of eleven address bits. Addressing the display this way would involve 2048 (2 to the eleventh power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- Map the 960 bytes of 40-column text into only 1024 bytes.
- Scan the low-order address to refresh the dynamic RAMs.
- Continue to refresh the RAMs during video blanking.

The requirements of the RAM refreshing are discussed earlier in this chapter in the section “Dynamic-RAM Refreshment.”

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals called S0, S1, S2, and S3, where S stands for sum. Figure 7-10 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5'. A constant value of 1 appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

Table 7-11. Display Address Transformation

			V3 Carry in
H5'	V3	H4	H3 Augend
V4	H5'	V4	1 Addend
S3	S2	S1	S0 Sum

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is 0 and the horizontal count is 24: H0, H1, H2, and H5 are 0's and H3, and H4 are 1's. The value of the sum is 0, so the memory location for the first character on the display is the first location in the display page, as you might expect.

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 7-12). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from 0 to 4 and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 7-12, so that rows 0 through 7 start on 127-byte boundaries. When the vertical row counter reaches 8, then V0, V1, and V2 are 0 again, and V3 changes to 1. If you do the addition in Table 7-11 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

Figure 7-10. 40-Column Text Display Memory

Memory locations marked with an asterisk () are reserved for use by peripheral I/O firmware: refer to the section "Peripheral-Card RAM Space" in Chapter 6.*

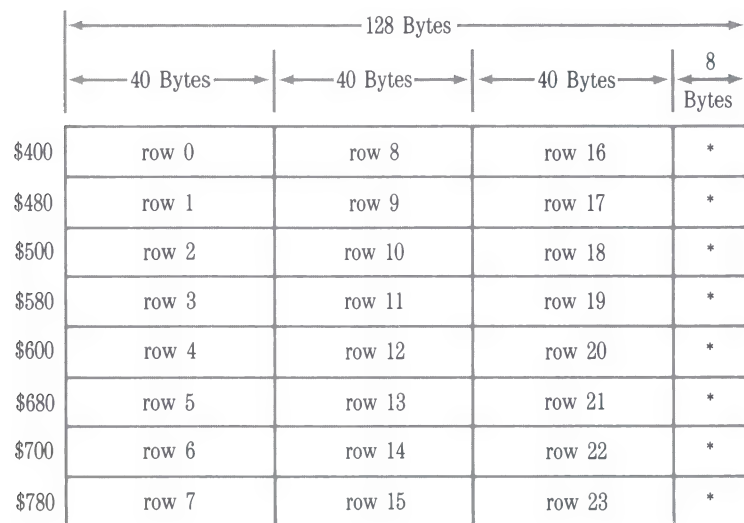


Figure 7-10 shows how groups of three forty-character rows are stored in blocks of 120 contiguous bytes starting on 127-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 2-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

Table 7-12 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2 and S3 are the folded address bits described above. Address bits marked with asterisks (*) are different for different modes: see Table 7-13 and the four subsections under the section “Video Display Modes.”

Table 7-12. Display Memory Addressing

Memory Address Bit	Display Address Bit	Memory Address Bit	Display Address Bit
A0	H0	A8	V1
A1	H1	A9	V2
A2	H2	A10	**
A3	S0	A11	**
A4	S1	A12	**
A5	S2	A13	**
A6	S3	A14	**
A7	V0	A15	GND

** For these address bits, see text and Table 7-13.

Table 7-13. Memory Address Bits for Display Modes

. means logical AND; ’ means logical NOT.

Address Bit	Text and Low-Resolution	Display Modes
		High-Resolution and Double-High-Resolution
A10	80STORE+PAGE2’	VA
A11	80STORE’.PAGE2	VB
A12	0	VC
A13	0	80STORE+PAGE2’
A14	0	80STORE’.PAGE2

Video Display Modes

The different display modes all use the address-mapping scheme described in the previous section, but they use different-sized memory areas in different locations. The next four sections describe the addressing schemes and the methods of generating the actual video signals for the different display modes.

Text Displays

The text and low-resolution graphics pages begin at memory locations \$0400 and \$0800. Table 7-13 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of PG2 and 80STORE, which are set by the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to 0. Notice that 80STORE active inhibits PG2: there is only one display page in 80-column mode.

The bit patterns used for generating the different characters are stored in a 32K ROM. The low-order six bits of each data byte reach the character generator ROM directly, via the video data bus VID0-VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator ROM on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator ROM along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator ROM puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator ROM are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit. The shift register is controlled by signals named LDPS' (for load parallel-to-serial shifter) and VID7M (for video 7 MHz). In 40-column mode, LDPS' strobes the output of the character generator ROM into the shift register once each microsecond, and bits are sent to the screen at a 7 MHz rate.

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory on the 80-column card are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0-VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously, but their outputs are sent to the character generator ROM alternately by $\phi 0$ and $\phi 1$. In 80-column mode, LDPS' loads data from the character generator ROM into the shift register twice during each microsecond, once during $\phi 0$ and once during $\phi 1$, and bits are sent to the screen at a 14 MHz rate. Figures 7-11a and 7-11b show the video timing signals.

Figure 7-11a. 7 MHz Video Timing Signals

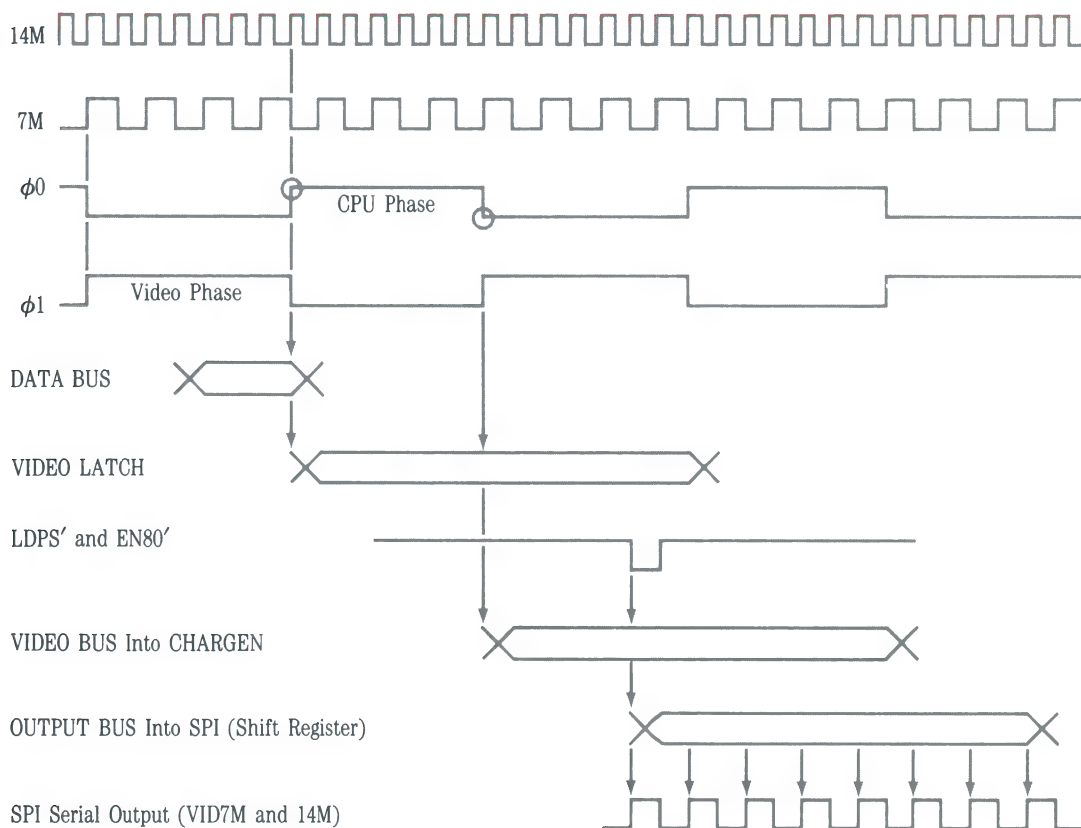
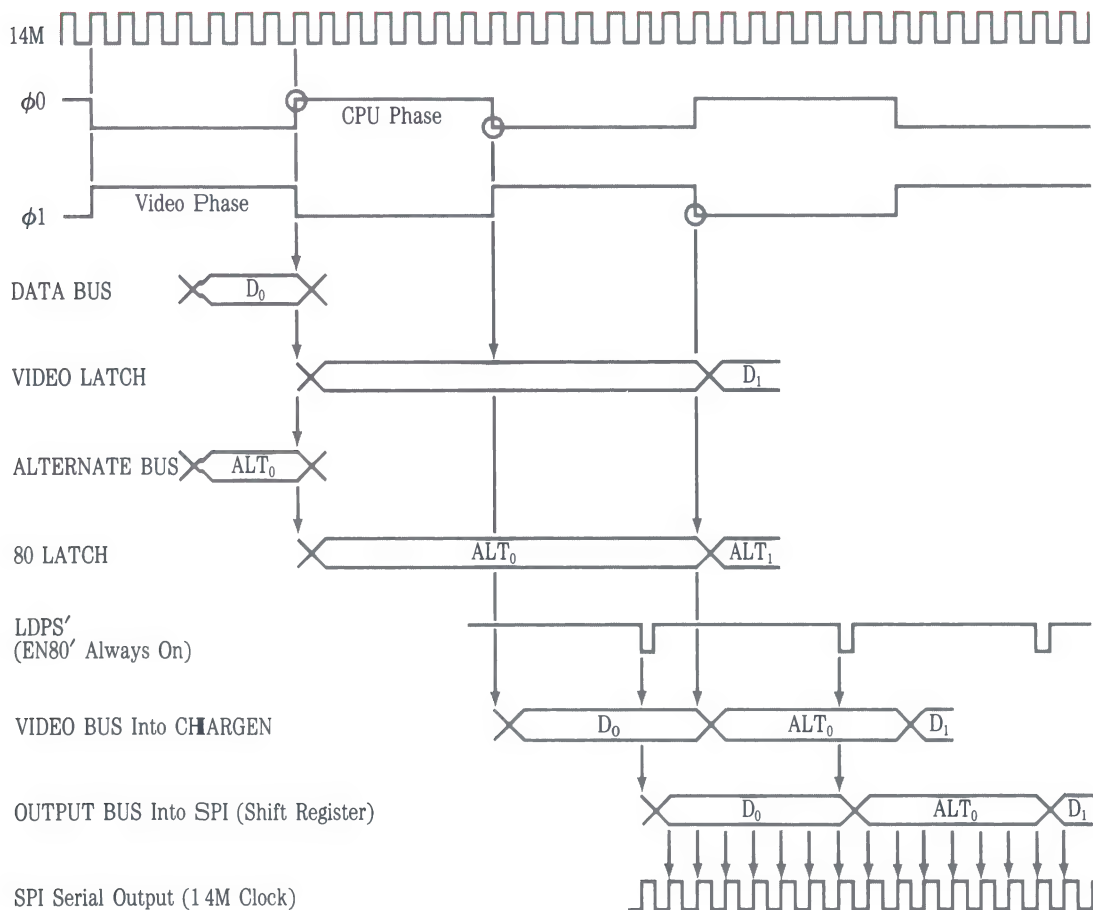


Figure 7-11b. 14 MHz Video Timing Signals



Low-Resolution Display

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIREŠ', as shown in Table 7-14.

Table 7-14. Character-Generator Control Signals

Display Mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIREŠ'	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nibbles. Each nibble selects one of 16 colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits.

The video signal generated by the Apple IIe includes a short burst of 3.58 MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58 MHz color signal. The Apple IIe's video signal produces color by interacting with this 3.58 MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58 MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1-MHz data clock. To generate a stream of fourteen bits from each eight-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same eight bits; the last two bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: .98 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58 MHz color signal, the phase relationship between the bit patterns and the signal changes by a half cycle for each successive pattern. To compensate for this, the character generator ROM puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

High-Resolution Display

The high-resolution graphics pages begin at memory locations \$2000 and \$4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PG2 and 80STORE, the signals controlled by the display-page (PAGE2) and 80-column-video (80COL) soft switches. As in text mode, 80STORE inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 7-10, but there are eight of these blocks. As Table 7-12 and Table 7-13 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display, VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block. It might help to think of it as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the eight bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0-VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58 MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58 MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating 1's and 0's

gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating 1's and 0's.

To produce different colors, the bit patterns must have different phase relationships to the 3.58 MHz color signal. If alternating 1's and 0's produce a certain color, say green, then reversing the pattern to 0's and 1's will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58 MHz color signal. In high-resolution mode, the Apple IIe produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the PAL device. If D7 is off, the PAL device transmits shift-register timing signals LDPS' and VID7M normally. If D7 is on, the PAL device delays LDPS' and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

A Note About Timing: For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the PAL device controls shift-register timing signals LDPS' and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.

Double-High-Resolution Display

Double-high-resolution graphics mode displays two bytes in the time normally required for one, but uses high-resolution graphics Page 1 in both main and auxiliary memory instead of text or low-resolution Page 1.

Note: There is a second pair of pages, high-resolution Page 2, which can be used to display a second double-high-resolution page.

Double-high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, seven from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appears in columns 0-6, 14-20, and so on, up to columns 547-552. Data from main memory appears in columns 7-13, 21-27, and so on, up to 553-559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth monitor (less than 14 MHz), single dots will be dimmer than normal.

Note: Except for some expensive RGB-type monitors, any video monitor with a bandwidth as high as 14 MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of $\phi 0$ clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch.

$\Phi 1$ ($\phi 1$) enables output from the (auxiliary) 80 latch, and $\phi 0$ enables output from the (main) video latch. Output from both latches goes to CHARGEN, where GR and SEGB' select high-resolution graphics. LDPS operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for double-high-resolution display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gate shift register then outputs to VID, the video display hybrid circuit, for output to the display device.

Video Output Signals

The stream of video data generated by the display circuits described above goes to a linear summing circuit built around transistor Q1 where it is mixed with the sync signals and the color burst. Resistors R3, R5, R7, R10, R13, and R15 adjust the signals to the proper amplitudes, and a tank circuit (L3 and C32) resonant at 3.58 MHz conditions the color burst.

The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video (see Table 7-15). This signal is available in two places in the Apple IIe:

- At the phono jack on the back of the Apple IIe. The sleeve of this jack is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms.
- At the internal video connector on the Apple IIe circuit board near the RCA jack, J13 in Figure 7-13c. It is made up of four Molex-type pins, 0.25 inches tall, on 0.10 inch centers. This connector carries the video signal, ground, and two power supplies, as shown in Table 7-15.

Table 7-15. Internal Video Connector Signals

Note: Pin 1 is the pin closest to the keyboard; pin 4 is at the back.

Pin	Name	Description
1	GROUND	System common ground
2	VIDEO	NTSC-compatible positive composite video. White level is about 2.0 volts, black level is about 0.75 volts, and sync level is 0.0 volts. This output is not protected against short-circuits.
3	-5V	-5 volt power supply
4	+12V	+12 volt power supply

Built-in I/O Circuits

The use of the Apple IIe's built-in I/O features is described in Chapter 2. This section describes the hardware implementation of all of those features except the video display described in the previous sections.

The IOU (Input/Output Unit) directly generates the output signals for the speaker, the cassette interface, and the annunciators. The other I/O features are handled by smaller ICs, as described later in this section.

The addresses of the built-in I/O features are described in Chapter 2 and listed in Table 2-2, Table 2-11, and Table 2-12. All of the built-in I/O features except the displays use memory locations between \$C000 and \$C070 (decimal 49152 and 49264). The I/O address decoding is performed by three ICs: a 74LS138, a 74LS154, and a 74LS251.

The 74LS138 decodes address lines A8, A9, A10, and A11 to select address pages on 256-byte boundaries starting at \$C000 (decimal 49152). When it detects addresses between \$C000 and \$C0FF, it enables the IOU and the 74LS154. The 74LS154 in turn decodes address lines A4, A5, A6, and A7 to select 16-byte address areas between \$C000 and \$C0FF. Addresses between \$C060 and \$C06F enable the 74LS251 that multiplexes the hand control switches and paddles; addresses between \$C070 and \$C07F reset the NE558 quadruple timer that interfaces to the hand controls, as described later in the section "Game I/O Signals."

The Keyboard

The Apple IIe's keyboard is a matrix of keyswitches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector. The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C70 and R32. The debounce time is also set externally, by C71.

The AY-3600's outputs include five bits of key code plus separate lines for **CONTROL**, **SHIFT**, any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code lines, along with **CONTROL** and **SHIFT**, are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named KBD' and ENKBD'. The KBD' signal is enabled by the MMU whenever a program reads location \$C000, as described in the section "Reading the Keyboard" in Chapter 2.

Table 7-16. Keyboard Connector Signals

Pin Number	Name	Description
1,2,4,6,8,10, 23,25,12,22	Y0-Y9	Y-direction key-matrix connections
3	+5	+5 volt supply
5,7,9,15	n.c.	
1	LCNTL'	Line from CONTROL key
13	GND	System common ground
14,16,20,21, 19,26,17	X0-X7	X-direction key-matrix connections
24	LSHFT'	Line from SHIFT key

Connecting a Keypad

There is a smaller connector wired in parallel with the keyboard connector. You can connect a ten-key numeric pad to the Apple IIe via this connector.

Table 7-17. Keypad Connector Signals

Pin Number	Name	Description
1,2,5,3,4,6	Y0-Y5	Y-direction key-matrix connections
7	n.c.	
9,11,10,8	X4-X7	X-direction key-matrix connections

Cassette I/O

The two miniature phone jacks on the back of the Apple IIe are used to connect an audio cassette recorder for saving programs. The output signal to the cassette recorder comes from a pin on the IOU via resistor network R6 and R9, which attenuates the signal to a level appropriate for the recorder's microphone input. Input from the recorder is amplified and conditioned by a type 741 operational amplifier and sent to one of the inputs of the 74LS251 input multiplexer.

The signal specifications for cassette I/O are

- Input: 1 volt (nominal) from recorder earphone or monitor output. Input impedance is 12K ohms.
- Output: 25 millivolts to recorder microphone input. Output impedance is 100 ohms.

The Speaker

The Apple IIe's built-in loudspeaker is controlled by a single bit of output from the IOU (Input Output Unit). The signal from the IOU is AC coupled to Q5, an MPSA13 Darlington transistor amplifier. The speaker connector is a Molex KK100 connector, J18 in Figure 7-13b, with two square pins 0.25 inches tall and on 0.10-inch centers.

A light-emitting diode is connected in parallel across the speaker pins such that, when the speaker is not connected, the diode glows whenever the speaker signal is on. This diode is used as a diagnostic indicator during assembly and testing of the Apple IIe.

Table 7-18. Speaker Connector Signals

Pin Number	Name	Description
1	SPKR	Speaker signal. This line will deliver about 0.5 watts into an 8-ohm speaker.
2	+5	+5V power supply. Note that the speaker is not connected to system ground.

Game I/O Signals

Several I/O signals that are individually controlled via soft switches are collectively referred to as the game signals. Even though they are normally used for hand controls, these signals can be used for other simple I/O applications. There are five output signals: the four annunciators, numbered A0 through A3, and one strobe output. There are three one-bit inputs, called *switches* and numbered SW0 through SW2, and four analog inputs, called *paddles* and numbered PDL0 through PDL3.

The annunciator outputs are driven directly by the IOU (Input Output Unit). These outputs can drive one TTL (transistor-transistor logic) load each; for heavier loads, you must use a transistor or a TTL buffer on these outputs. These signals are only available on the 16-pin internal connector. (See Table 7-19.)

The strobe output is a pulse transmitted any time a program reads or writes to location \$C040. The strobe pin is connected to one output of the 74LS154 address decoder. This TTL signal is normally high; it goes low during $\phi 0$ of the instruction cycle that addresses location \$C040. This signal is only available on the 16-pin internal connector. (See Table 7-19.)

The game inputs are multiplexed along with the cassette input signal by a 74LS251 eight-input multiplexer enabled by the C06X' signal from the 74LS154 I/O address decoder. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus.

The switch inputs are standard low-power Schottky TTL inputs. To use them, connect each one to 560-ohm pull-down resistors connected to the ground and through single-pole, momentary-contact pushbutton switches to the +5 volt supply.

The hand-control inputs are connected to the timing inputs of an NE558 quadruple 555-type analog timer. Addressing \$C07X sends a signal from the 74LS154 that resets all four timers and causes their outputs to go to 1 (high). A variable resistance of up to 150K ohms connected between one of these inputs and the +5V supply controls the charging time of one of four 0.022-microfarad capacitors. When the voltage on the capacitor passes a certain threshold, the output of the NE558 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.

The game I/O signals are all available on a 16-pin DIP socket labelled GAME I/O on the main circuit board inside the case. The switches and the paddles are also available on a D-type miniature connector on the back of the Apple IIe; see J8 and J15 in Figure 7-13d.

Table 7-19. Game I/O Connector Signals

Internal-Connector Pin Number	Back-Panel-Connector Pin Number	Signal Name	Description
1	2	+5V	+5V power supply. Total current drain from this pin must not exceed 100mA.
2,3,4	7,1,6	PB0-PB2	Switch inputs. These are standard 74LS inputs.
5		STROBE'	Strobe output. This line goes low during ϕ_0 of a read or write instruction to location \$C040.
6,10,7,11	5,8,4,9	PDL0-PDL3	Hand control inputs. Each of these should be connected to a 150K-ohm variable resistor connected to +5V.
8	3	GND	System ground.
15,14,13,12	-	AN0-AN3	Annunciators. These are standard 74LS TTL outputs and must be buffered to drive other than TTL inputs.
9,16		n.c.	Nothing is connected to these pins.

Expanding the Apple IIe

Chapter 6 describes the standards for programming peripheral cards for the Apple IIe.

The main circuit board of the Apple IIe has eight empty card connectors or slots on it. These slots make it possible to add features to the Apple IIe by plugging in peripheral cards with additional hardware. This section describes the hardware that supports them, including all of the signals available on the expansion slots.

The Expansion Slots

The seven connectors lined up across the back part of the Apple IIe's main circuit card are the expansion slots, also called peripheral slots or simply slots, numbered from 1 to 7. They are 50-pin PC-card edge connectors with pins on 0.10-inch centers. A PC card plugged into one of these connectors has access to all of the signals necessary to perform input and output and to execute programs in RAM or ROM on the card. These signals are described briefly in Table 7-20. The following paragraphs describe the signals in general and mention a few points that are often overlooked. For further details, refer to the schematic diagram in Figures 7-13a, 7-13b, 7-13c, and 7-13d.

The Peripheral Address Bus

The microprocessor's address bus is buffered by two 74LS244 octal three-state buffers. These buffers, along with a buffer in the microprocessor's R/W' line, are enabled by a signal derived from the DMA' daisy-chain on the expansion slots. Pulling the peripheral line DMA' low disables the address and R/W' buffers so that peripheral DMA circuitry can control the address bus. The DMA address and R/W' signals supplied by a peripheral card must be stable all during ϕ_0 of the instruction cycle, as shown in Figure 7-12.

Another signal that can be used to disable normal operation of the Apple IIe is INH'. Pulling INH' low disables all of the memory in the Apple IIe except the part in the I/O space from \$C000 to \$CFFF. A peripheral card that uses either INH' or DMA' must observe proper timing; in order to disable RAM and ROM cleanly, the disabling signal must be stable all during ϕ_0 of the instruction cycle (refer to the timing diagram in Figure 7-12).

The peripheral devices should use I/O SELECT' and DEVICE SELECT' as enables. Most peripheral ICs require their enable signals to be present for a certain length of time before data is strobed into or out of the device. Remember that I/O SELECT' and DEVICE SELECT' are only asserted during $\phi 0$ high.

The Peripheral Data Bus

The Apple IIe has two versions of the microprocessor data bus: an internal bus, MD0-MD7, connected directly to the microprocessor; and an external bus, D0-D7, driven by a 74LS245 octal bidirectional bus buffer. The 65C02 is fabricated with MOS circuitry, so it can drive capacitive loads of up to about 130 pF. If peripheral cards are installed in all seven slots, the loading on the data bus can be as high as 500 pF, so the 74LS245 drives the data bus for the peripheral cards. The same argument applies if you use MOS devices on peripheral cards: they don't have enough drive for the fully-loaded bus, so you should add buffers.

Loading and Driving Rules

Table 7-20 shows the drive requirements and loading limits for each pin on the expansion slots. The address bus, the data bus, and the R/W' line should be driven by three-state buffers. Remember that there is considerable distributed capacitance on these busses and that you should plan on tolerating the added load of up to six additional peripheral cards. MOS devices such as PIAs and ACIAs cannot switch such heavy capacitive loads. Connecting such devices directly to the bus will lead to possible timing and level errors.

Interrupt and DMA Daisy Chains

The interrupt requests (IRQ' and NMI') and the direct-memory access (DMA') signal are available at all seven expansion slots. A peripheral card requests an interrupt or a DMA transfer by pulling the appropriate output line low (active). If two peripheral cards request an interrupt or a DMA transfer at the same time, they will contend for the data and address busses. To prevent this, two pairs of pins on each connector are wired as a priority daisy chain. The daisy-chain pins for interrupts are INT IN and INT OUT, and the pins for DMA are DMA IN and DMA OUT, as shown for J1-J7 in Figure 7-13d.

Each daisy chain works like this: the output from each connector goes to the input of the next higher numbered one. For these signals to be useful for cards in lower numbered connectors, all of the higher numbered connectors must have cards in them, and all of those cards must connect DMA IN to DMA OUT and INT IN to INT OUT. Whenever a peripheral card uses pin DMA', it must do so only if its DMA IN line is active, and it must disable its DMA OUT line while it is using DMA'. The INT IN and INT OUT lines must be used the same way: enable the card's interrupt circuits with INT IN, and disable INT OUT whenever IRQ' or NMI' is being used.

Figure 7-12. Peripheral-Signal Timing

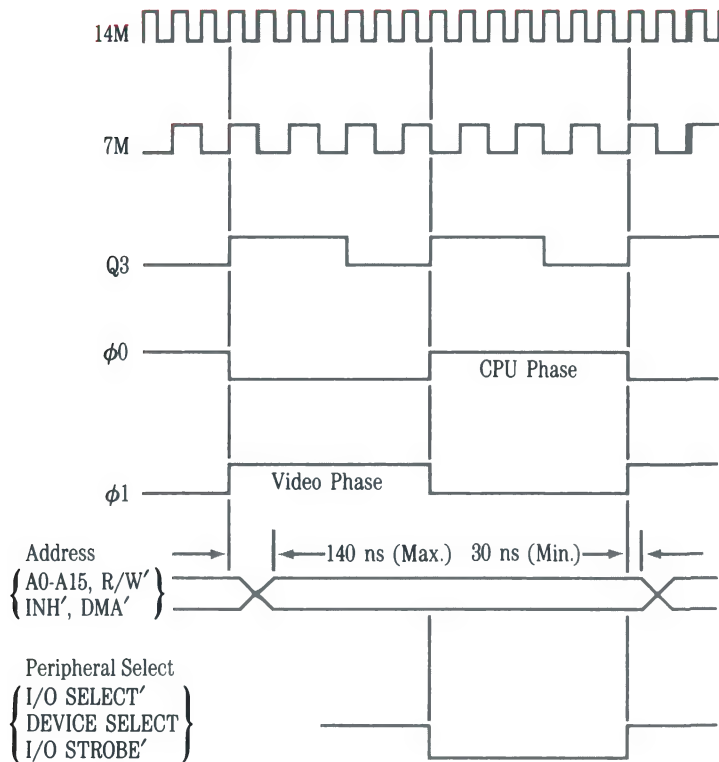


Table 7-20. Expansion Slot Signals

Pin	Name	Description
1	I/O SELECT	Normally high; goes low during $\phi 0$ when the 65C02 addresses location \$CnXX, where n is the connector number. This line can drive 10 LS TTL loads.*
2-17	A0-A15	Three-state address bus. The address becomes valid during $\phi 1$ and remains valid during $\phi 0$. Each address line can drive 5 LS TTL loads.*
18	R/W'	Three-state read/write line. Valid at the same time as the address bus; high during a read cycle, low during a write cycle. It can drive 2 LS TTL loads.*
19	SYNC'	Composite horizontal and vertical sync, on expansion slot 7 <i>only</i> . This line can drive 2 LS TTL loads.*
20	I/O STROBE'	Normally high; goes low during $\phi 0$ when the 65C02 addresses a location between \$C800 and \$CFFF. This line can drive 4 LS TTL loads.
21	RDY	Input to the 65C02. Pulling this line low during $\phi 1$ halts the 65C02 with the address bus holding the address of the location currently being fetched. This line has a 3300 ohm pullup resistor to +5V.
22	DMA'	Input to the address bus buffers. Pulling this line low during $\phi 1$ disconnects the 65C02 from the address bus. This line has a 3300 ohm pullup resistor to +5V.
23	INT OUT	Interrupt priority daisy-chain output. Usually connected to pin 28 (INT IN).†
24	DMA OUT	DMA priority daisy-chain output. Usually connected to pin 22 (DMA IN).
25	+5V	+5-volt power supply. A total of 500mA is available for all peripheral cards.
26	GND	System common ground.
27	DMA IN	DMA priority daisy-chain input. Usually connected to pin 24 (DMA OUT).
28	INT IN	Interrupt priority daisy-chain input. Usually connected to pin 23 (INT OUT).
29	NMI'	Non-maskable interrupt to 65C02. Pulling this line low starts an interrupt cycle with the interrupt-handling routine at location \$03FB. This line has a 3300 ohm pullup resistor to +5V.

Table 7-20—Continued. Expansion Slot Signals

Pin	Name	Description
30	IRQ'	Interrupt request to 65C02. Pulling this line low starts an interrupt cycle only if the interrupt-disable (I) flag in the 65C02 is not set. Uses the interrupt-handling routine at location \$03FE. This line has a 3300 ohm pullup resistor to +5V.
31	RES'	Pulling this line low initiates a reset routine, as described in Chapter 4.
32	INH'	Pulling this line low during $\phi 1$ inhibits (disables) the memory on the main circuit board. This line has a 3300 ohm pullup resistor to +5V.
33	-12V	-12 volt power supply. A total of 200mA is available for all peripheral cards.
34	-5V	-5 volt power supply. A total of 200mA is available for all peripheral cards.
35	3.58M	3.58 MHz color reference signal, on slot 7 <i>only</i> . This line can drive 2 LS TTL loads.*
36	7M	System 7 MHz clock. This line can drive 2 LS TTL loads.*
37	Q3	System 2 MHz asymmetrical clock. This line can drive 2 LS TTL loads.*
38	$\phi 1$	65C02 phase 1 clock. This line can drive 2 LS TTL loads.*
39	μ PSYNC	The 65C02 signals an operand fetch by driving this line high during the first read cycle of each instruction.
40	$\phi 0$	65C02 phase 0 clock. This line can drive 2 LS TTL loads.*
41	DEVICE SELECT'	Normally high; goes low during $\phi 0$ when the 65C02 addresses location \$C0nX, where n is the connector number plus 8. This line can drive 10 LS TTL loads.*
42-49	D0-D7	Three-state buffered bi-directional data bus. Data becomes valid during $\phi 0$ high and remains valid until $\phi 0$ goes low. Each data line can drive one LS TTL load.*
50	+12V	+12 volt power supply. A total of 250mA is available for all peripheral cards.

* Loading limits are for each card.

† On slot 7 *only*, this pin can be connected to the graphics-mode signal GR: see text for details.

Auxiliary Slot

The large connector at the left side of the Apple IIe's main circuit card is the auxiliary slot. It is a 60-pin PC-card edge connector with pins on 0.10-inch centers. A PC card plugged into this connector has access to all of the signals used in producing the video display. These signals are described briefly in Table 7-21. For further details, refer to the schematic diagram in Figures 7-13a, 7-13b, 7-13c, and 7-13d.

Many of the internal signals that are not available on the expansion slots are on the auxiliary slot. By using both kinds of connectors, manufacturing and repair personnel can gain access to most of the signals needed for diagnosing problems in the Apple IIe.

80-Column Display Signals

The additional memory needed for producing an 80-column text display is on the 80-column text card, along with the buffers that transfer the data to the video data bus, as described earlier in this chapter in the section "Text Displays." The signals that control the 80-column text data include the system clocks $\phi 0$ and $\phi 1$, the multiplexed RAM address RA0-RA7, the RAM address-strobe signals PRAS' and PCAS', and the auxiliary-RAM enable signals, EN80' and R/W80. The EN80' enable signal is controlled by the 80STORE soft switch described in Chapter 4. Data is sent to the auxiliary memory via the internal data bus MD0-MD7; the data is transferred to the video generator via the video data bus VID0-VID7.

Table 7-21. Auxiliary Slot Signals

Pin	Name	Description
1	3.58M	3.58 MHz video color reference signal. This line can drive two LS TTL loads.
2	VID7M	Clocks the video dots out of the 74166 parallel-to-serial shift register. This line can drive two LS TTL loads.
3	SYNC'	Video horizontal and vertical sync signal. This line can drive two LS TTL loads.
4	PRAS'	Multiplexed RAM row-address strobe. This line can drive two LS TTL loads.
5	VC	Third low-order vertical-counter bit. This line can drive two LS TTL loads.
6	C07X'	Hand-control reset signal. This line can drive two LS TTL loads.
7	WNDW'	Video non-blank window. This line can drive two LS TTL loads.
8	SEGA	First low-order vertical counter bit. This line can drive two LS TTL loads.
51,10,49,48, 13,14,46,9	RA0-RA7	Multiplexed RAM-address bus. This line can drive two LS TTL loads.
11,12	ROMEN1, ROMEN2	Enable signals for the ROMs on main circuit board.
44,43,40,39, 21,20,17,16	MD0-MD7	Internal (unbuffered) data bus. This line can drive two LS TTL loads.
45,42,41,38, 22,19,18,15	VID0-VID7	Video data bus. This three-state bus carries video data to the character generator.
23	$\phi 0$	65C02 clock phase 0. This line can drive two LS TTL loads.
24	CLRGAT'	Color-burst gating signal. This line can drive two LS TTL loads.
25	80VID'	Enables 80-column display timing. This line can drive two LS TTL loads.
26	EN80'	Enable for auxiliary RAM. This line can drive two LS TTL loads.
27	ALTVID'	Alternative video output to the video summing amplifier.
28	SEROUT'	Video serial output from 74166 parallel-to-serial shift register.
29	ENVID'	Normally low; driving this line high disables the character generator such that the video dots from the shift register are all high (white), and alternative video can be sent out via ALTVID'. This line has a 1000 ohm pulldown resistor to ground.

Table 7-21—Continued. Auxiliary Slot Signals

Pin	Name	Description
30	+5	+5 volt power supply.
31	GND	System common ground.
32	14M	14.3 MHz master clock signal. This line can drive two LS TTL loads.
33	PCAS'	Multiplexed column-address strobe. This line can drive two LS TTL loads.
34	LDPS'	Strobe to video parallel-to-serial shift register. This signal goes low to load the contents of the video data bus into the shift register. This line can drive two LS TTL loads.
35	R/W80	Read/write signal for RAM on the card in this slot. This line can drive two LS TTL loads.
36	$\phi 1$	65C02 clock phase 1. This line can drive two LS TTL loads.
37	CASEN'	Column-address enable. This signal is disabled (held high) during accesses to memory on the card in this slot. This line can drive two LS TTL loads.
47	H0	Low-order horizontal byte counter. This line can drive two LS TTL loads.
50	AN3	Output of annunciator number 3. This line can drive two LS TTL loads.
52	R/W'	65C02 read/write signal. This line can drive two LS TTL loads.
53	Q3	2 MHz asymmetrical clock. This line can drive two LS TTL loads.
54	SEGB	Second low-order vertical-counter bit. This line can drive two LS TTL loads.
55	FRCTXT'	Normally high; pulling this line low enables 14MHz video output even when GR is active.
56,57	RA9',RA10'	Character-generator control signals from the IOU. This line can drive two LS TTL loads.
58	GR	Graphics-mode enable signal. This line can drive two LS TTL loads.
59	7M	7 MHz timing signal. This line can drive two LS TTL loads.
60	ENTMG'	Normally low; pulling this line high disables the master timing from the PAL device. This line has a 1000 ohm pulldown resistor to ground.

Figure 7-13a. Schematic Diagram, Part 1

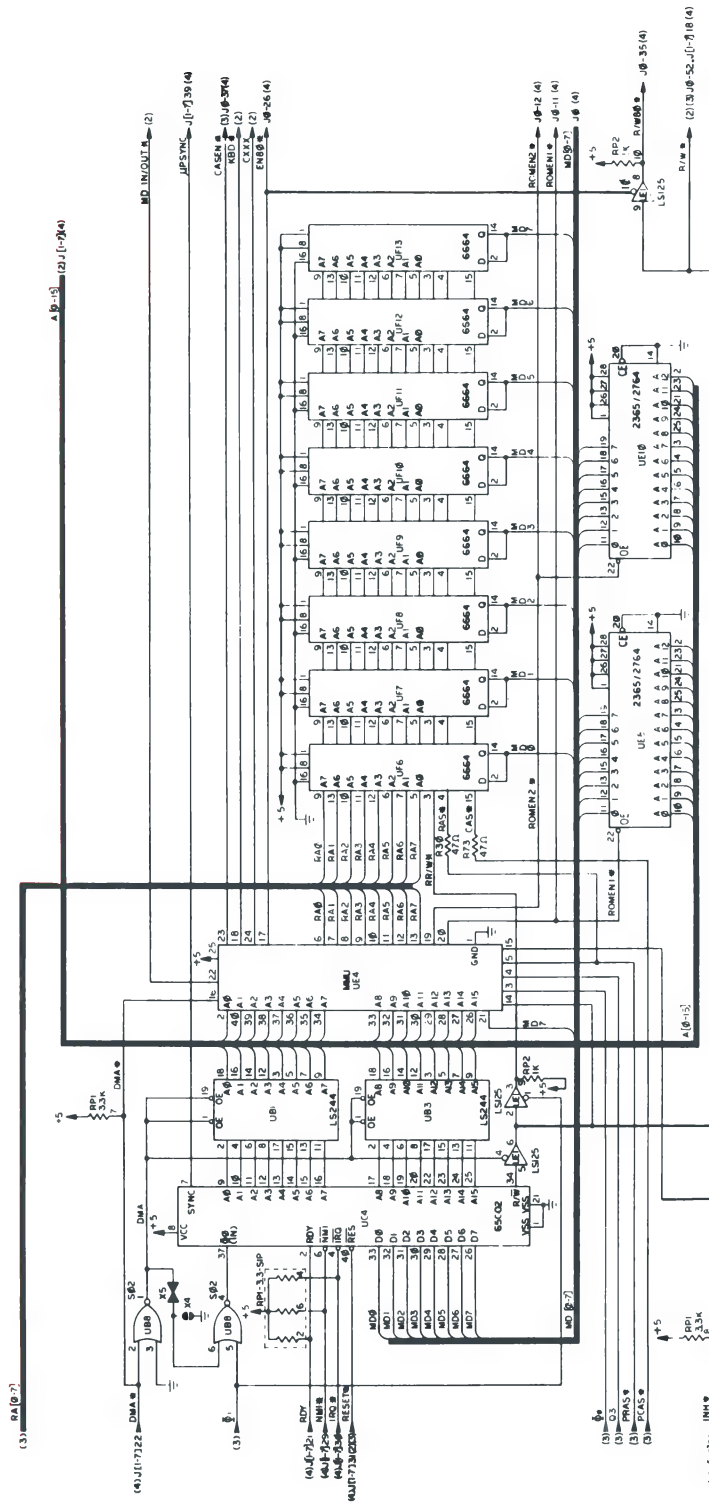
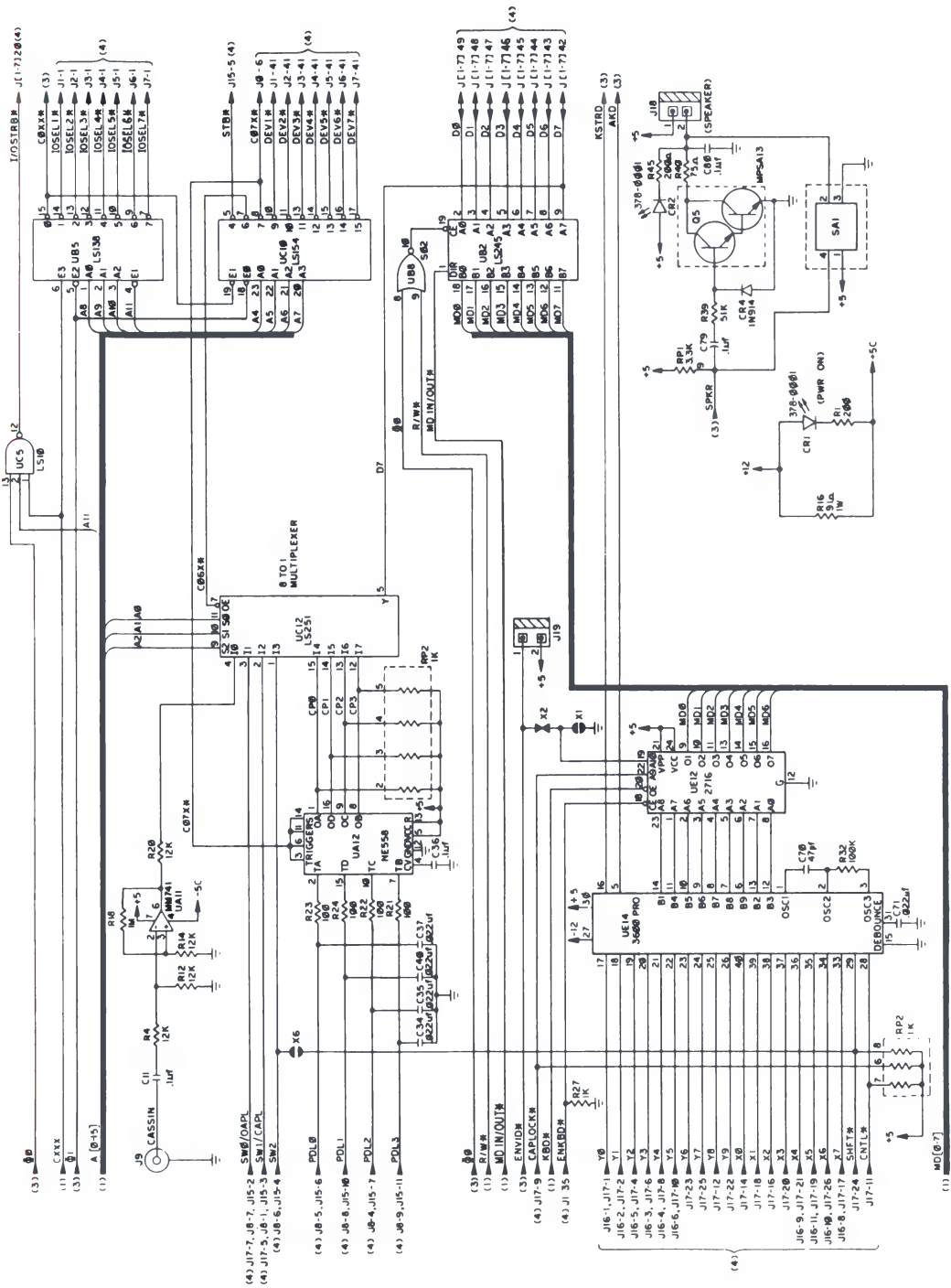


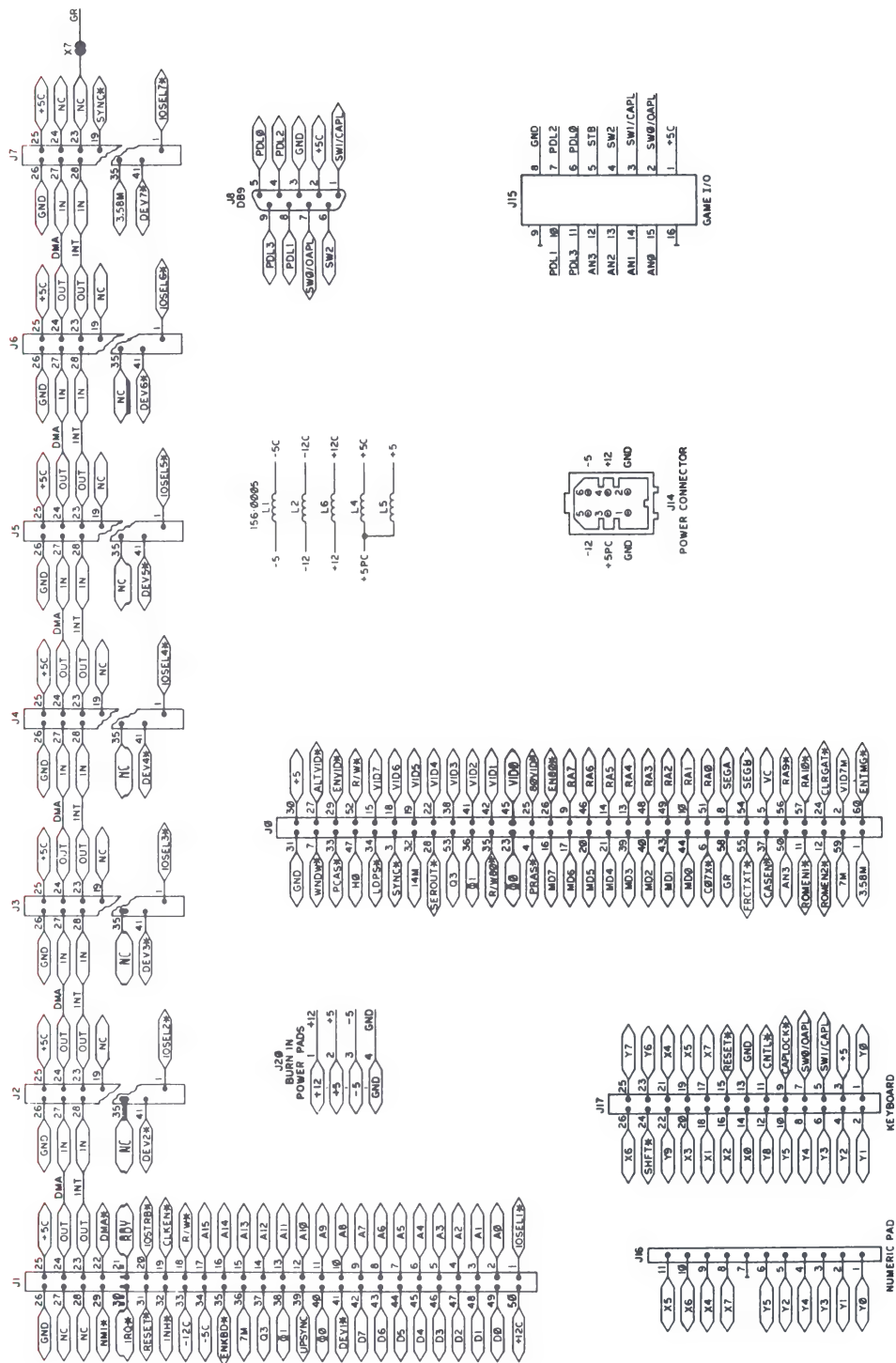
Figure 7.13b. Schematic Diagram, Part 2

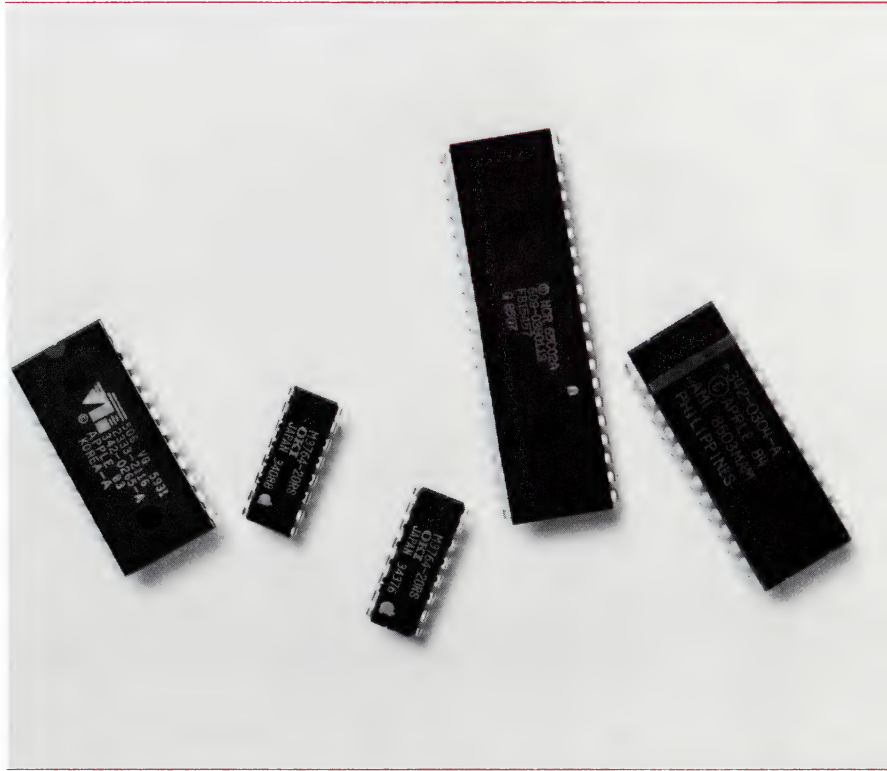


202



Figure 7-14d. Schematic Diagram. Part 4





This appendix contains a description of the differences between the 6502 and the 65C02 microprocessors. It also contains the data sheet for the 65C02 microprocessor.

The 6502 microprocessor was used in the original Apple IIe, Apple II Plus, and Apple II. The 65C02 is a 6502 that uses less power and has ten new instructions and two new addressing modes. The 65C02 is used in both the enhanced Apple IIe and the Apple IIc.

In the data sheet tables, execution times are specified in number of cycles. One cycle time for the Apple IIe equals 0.978 microseconds, giving a system clock rate of about 1.02 MHz.

Note: If you want to write programs that execute on all computers in the Apple II series, use only those 65C02 instructions that are also present on the 6502.

Differences Between 6502 and 65C02

The data sheet lists the instructions and addressing modes of the 65C02. This section supplements that information by listing those instructions whose execution times or results differ in the 6502 and the 65C02.

Different Cycle Times

A few instructions on the 65C02 operate in different numbers of cycles than their 6502 equivalents. These instructions are listed in Table A-1.

Table A-1. Cycle Time Differences

Instruction/Mode	Opcode	6502 Cycles	65C02 Cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

Different Instruction Results

It is important to note that the BIT instruction when used in immediate mode (opcode \$89) leaves processor status register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the status register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6.

Also note that if the JMP indirect instruction (code \$6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP (\$02FF) gets ADL from location \$02FF on both processors. But on the 65C02, ADH comes from \$0300; on the 6502, ADH comes from \$0200.

Data Sheet

The remaining pages of this appendix are copyright 1982, NCR Corporation, Dayton, Ohio, and are reprinted with their permission.

■ GENERAL DESCRIPTION

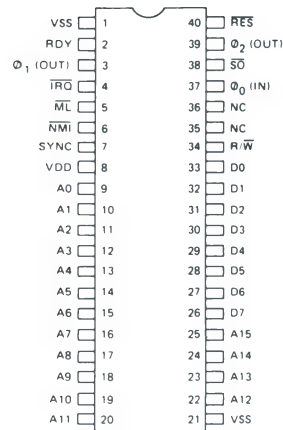
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

■ FEATURES

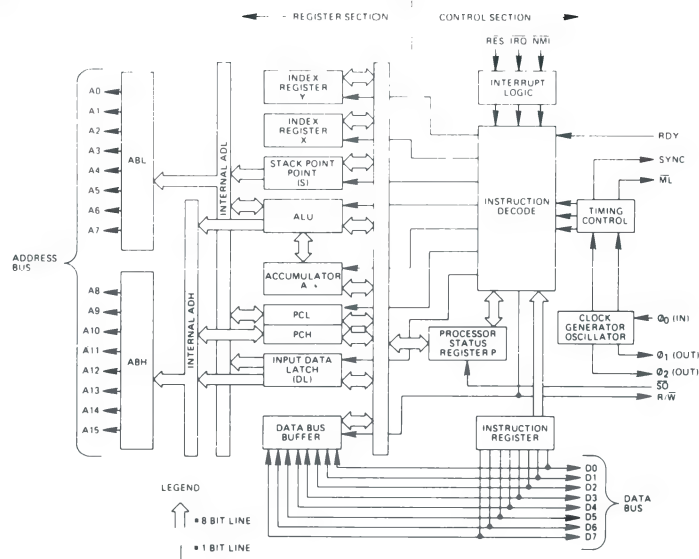
- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 HZ for even lower power consumption (pseudo-static: stop during Φ_2 high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ, $\overline{S0}$, NMI and RES)

* Specifications are subject to change without notice.

■ PIN CONFIGURATION



■ NCR65C02 BLOCK DIAGRAM



NCR65C02**■ ABSOLUTE MAXIMUM RATINGS:** ($V_{DD} = 5.0\text{ V} \pm 5\%$, $V_{SS} = 0\text{ V}$, $T_A = 0^\circ\text{ to } +70^\circ\text{C}$)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	-0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	-0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to +70	°C
STORAGE TEMP.	T_{STG}	-55 to +150	°C

■ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
\overline{IRQ}^*	Interrupt Request
RDY [*]	Ready
ML	Memory Lock
\overline{NMI}^*	Non-Maskable Interrupt
SYNC	Synchronize
RES [*]	Reset
SO [*]	Set Overflow
NC	No Connection
R/W	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
\emptyset_0	Clock Input
\emptyset_1, \emptyset_2	Clock Output

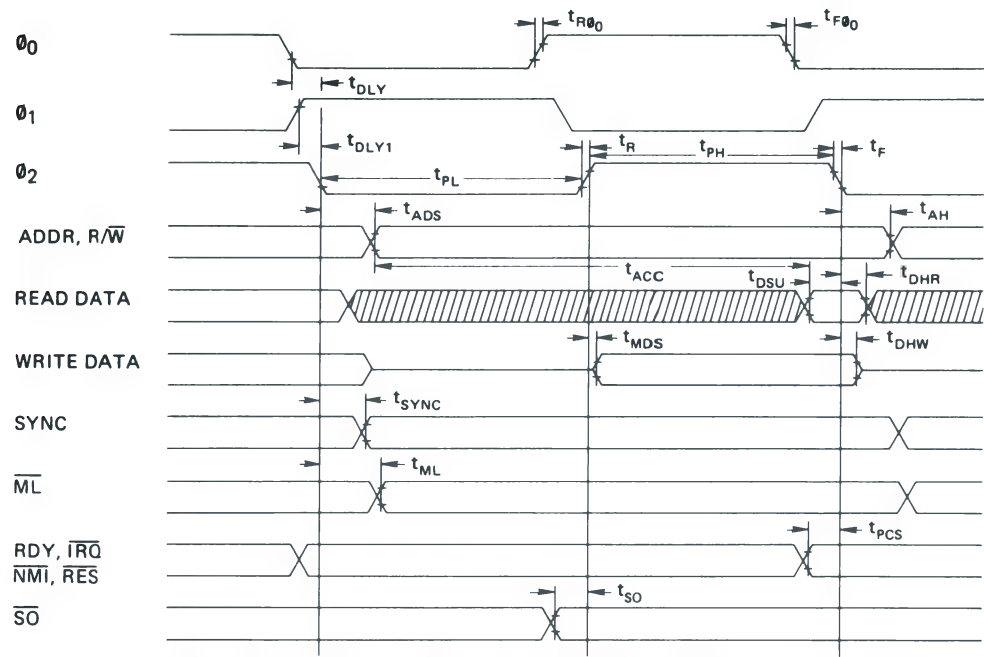
*This pin has an optional internal pullup for a No Connect condition.

■ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage \emptyset_0 (IN)	V_{IH}	$V_{SS} + 2.4$	—	V_{DD}	V
Input High Voltage RES, \overline{NMI} , RDY, \overline{IRQ} , Data, S.O.		$V_{SS} + 2.0$	—	—	V
Input Low Voltage \emptyset_0 (IN)	V_{IL}	$V_{SS} - 0.3$	—	$V_{SS} + 0.4$	V
RES, \overline{NMI} , RDY, \overline{IRQ} , Data, S.O.		—	—	$V_{SS} + 0.8$	V
Input Leakage Current ($V_{IN} = 0$ to 5.25V, $V_{DD} = 5.25\text{V}$)	I_{IN}				
With pullups		-30	—	+30	μA
Without pullups		—	—	+1.0	μA
Three State (Off State) Input Current ($V_{IN} = 0.4$ to 2.4V, $V_{CC} = 5.25\text{V}$)					
Data Lines	I_{TSI}	—	—	10	μA
Output High Voltage ($I_{OH} = -100\ \mu\text{A}$, $V_{DD} = 4.75\text{V}$)					
SYNC, Data, A0-A15, R/W	V_{OH}	$V_{SS} + 2.4$	—	—	V
Out Low Voltage ($I_{OL} = 1.6\text{mA}$, $V_{DD} = 4.75\text{V}$)					
SYNC, Data, A0-A15, R/W	V_{OL}	—	—	$V_{SS} + 0.4$	V
Supply Current $f = 1\text{MHz}$	I_{DD}	—	—	4	mA
Supply Current $f = 2\text{MHz}$	I_{DD}	—	—	8	mA
Capacitance ($V_{IN} = 0$, $T_A = 25^\circ\text{C}$, $f = 1\text{MHz}$)	C				pF
Logic	C_{IN}	—	—	5	
Data		—	—	10	
A0-A15, R/W, SYNC	C_{out}	—	—	10	
\emptyset_0 (IN)	C_{\emptyset_0} (IN)	—	—	10	

NCR65C02

■ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

■ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG, X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

■ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND, X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

■ MICROPROCESSOR PROGRAMMING MODEL



■ FUNCTIONAL DESCRIPTION

Timing Control

The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

Program Counter

The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

Instruction Register and Decode

Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

Accumulator

The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

Index Registers

There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

Stack Pointer

The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts (NMI and IRQ). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

Processor Status Register

The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

■ AC CHARACTERISTICS $V_{DD} = 5.0V \pm 5\%$, $T_A = 0^\circ C$ to $70^\circ C$, Load = 1 TTL + 130 pF

Parameter	Symbol	1MHz		2MHz		3MHz		Unit
		Min	Max	Min	Max	Min	Max	
Delay Time, θ_0 (IN) to θ_2 (OUT)	t_{DLY}	—	60	—	60	20	60	nS
Delay Time, θ_1 (OUT) to θ_2 (OUT)	t_{DLY1}	—20	20	—20	20	—20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	μS
Clock Pulse Width Low	t_{PL}	460	—	220	—	160	—	nS
Clock Pulse Width High	t_{PH}	460	—	220	—	160	—	nS
Fall Time, Rise Time	t_F, t_R	—	25	—	25	—	25	nS
Address Hold Time	t_{AH}	20	—	20	—	0	—	nS
Address Setup Time	t_{ADS}	—	225	—	140	—	110	nS
Access Time	t_{ACC}	650	—	310	—	170	—	nS
Read Data Hold Time	t_{DHR}	10	—	10	—	10	—	nS
Read Data Setup Time	t_{DSU}	100	—	60	—	60	—	nS
Write Data Delay Time	t_{MDS}	—	30	—	30	—	30	nS
Write Data Hold Time	t_{DHW}	20	—	20	—	15	—	nS
\overline{SO} Setup Time	t_{SO}	100	—	100	—	100	—	nS
Processor Control Setup Time**	t_{PCS}	200	—	150	—	150	—	nS
SYNC Setup Time	t_{SYNC}	—	225	—	140	—	100	nS
ML Setup Time	t_{ML}	—	225	—	140	—	100	nS
Input Clock Rise/Fall Time	$t_{F\theta_0}, t_{R\theta_0}$	—	25	—	25	—	25	nS

*NCR65C02 can be held static with θ_2 high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

■ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code Bytes Cycles X2 2 2 X3, X7, XB, XF 1 1 44 2 3 54, D4, F4 2 4 5C 3 8 DC, FC 3 4
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

■ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during θ_2 .
Unused input-only pins (\overline{IRQ} , \overline{NMI} , RDY, \overline{RES} , \overline{SO}).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

Implied Addressing [Implied]

In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]

This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]

With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

Absolute Addressing [Absolute]

For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]

Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]

Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]

Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]

Relative addressing is used only with branch instructions;

it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is -128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]

With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**

With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]

This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**

In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)

The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

■ SIGNAL DESCRIPTION

Address Bus (A0-A15)

A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks (θ_0 , θ_1 , and θ_2)

θ_0 is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The θ_2 clock output is in phase with θ_0 . The θ_1 output pin is 180° out of phase with θ_0 . (See timing diagram.)

Data Bus (D0-D7)

The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{\text{IRQ}}$)

This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{\text{IRQ}}$ is sampled during θ_2 operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during θ_1 . The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{\text{IRQ}}$ s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{\text{ML}}$)

In a multiprocessor system, the $\overline{\text{ML}}$ output indicates the need to defer the arbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{\text{ML}}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{\text{NMI}}$)

A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{\text{NMI}}$ is sampled during θ_2 ; the current instruction is completed and the interrupt sequence begins during θ_1 . The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{\text{NMI}}$ can occur before the first is finished. Care should be taken when using $\overline{\text{NMI}}$ to avoid this.

Ready (RDY)

This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one (θ_1), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (θ_2) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{\text{RES}}$)

This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after VDD reaches operating voltage from a power down. A positive transition on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{\text{RES}}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write ($\overline{\text{R/W}}$)

This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{\text{SO}}$)

A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of θ_1 .

Synchronize (SYNC)

This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during θ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the θ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

NCR65C02

■ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC	Add Memory to Accumulator with Carry	LDX	Load Index X with Memory
AND	"AND" Memory with Accumulator	LDY	Load Index Y with Memory
ASL	Shift One Bit Left	LSR	Shift One Bit Right
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set	ORA	"OR" Memory with Accumulator
BEQ	Branch on Result Zero	PHA	Push Accumulator on Stack
BIT	Test Memory Bits with Accumulator	PHP	Push Processor Status on Stack
BMI	Branch on Result Minus	*PHX	Push Index X on Stack
BNE	Branch on Result not Zero	*PHY	Push Index Y on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
*BRA	Branch Always	PLP	Pull Processor Status from Stack
BRK	Force Break	*PLX	Pull Index X from Stack
BVC	Branch on Overflow Clear	*PLY	Pull Index Y from Stack
BVS	Branch on Overflow Set	ROL	Rotate One Bit Left
CLC	Clear Carry Flag	ROR	Rotate One Bit Right
CLD	Clear Decimal Mode	RTI	Return from Interrupt
CLI	Clear Interrupt Disable Bit	RTS	Return from Subroutine
CLV	Clear Overflow Flag	SBC	Subtract Memory from Accumulator with Borrow
CMP	Compare Memory and Accumulator	SEC	Set Carry Flag
CPX	Compare Memory and Index X	SED	Set Decimal Mode
CPY	Compare Memory and Index Y	SEI	Set Interrupt Disable Bit
*DEA	Decrement Accumulator	STA	Store Accumulator in Memory
DEC	Decrement by One	STX	Store Index X in Memory
DEX	Decrement Index X by One	STY	Store Index Y in Memory
DEY	Decrement Index Y by One	*STZ	Store Zero in Memory
EOR	"Exclusive-or" Memory with Accumulator	TAX	Transfer Accumulator to Index X
*INA	Increment Accumulator	TAY	Transfer Accumulator to Index Y
INC	Increment by One	*TRB	Test and Reset Memory Bits with Accumulator
INX	Increment Index X by One	*TSB	Test and Set Memory Bits with Accumulator
INY	Increment Index Y by One	TSX	Transfer Stack Pointer to Index X
JMP	Jump to New Location	TXA	Transfer Index X to Accumulator
JSR	Jump to New Location Saving Return Address	TXS	Transfer Index X to Stack Pointer
LDA	Load Accumulator with Memory	TYA	Transfer Index Y to Accumulator

Note: * = New Instruction

■ MICROPROCESSOR OP CODE TABLE

S	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs			0
1	BPL	ORA rel	ORA*† (zpg)		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X			1
2	JSR	AND abs	AND ind, X		BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs			2
3	BMI	AND rel	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X			3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs			4
5	BVC	EOR rel	EOR*† (zpg)			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X			5
6	RTS	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs			6
7	BVS	ADC rel	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X			7
8	BRA*	STA rel	STA ind, X		STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs			8
9	BCC	STA rel	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X			9
A	LDY	LDA imm	LDA ind, X		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs			A
B	BCS	LDA rel	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y			B
C	CPY	CMP imm	CMP ind, X		CPY zpg	CMP zpg	DEC zpg		INY	CMP zpg	DEX		CPY abs	CMP abs	DEC abs			C
D	BNE	CMP rel	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X			D
E	CPX	SBC imm	SBC ind, X		CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs			E
F	BEQ	SBC rel	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC zpg	PLX*			SBC abs, X	INC abs, X			F
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes

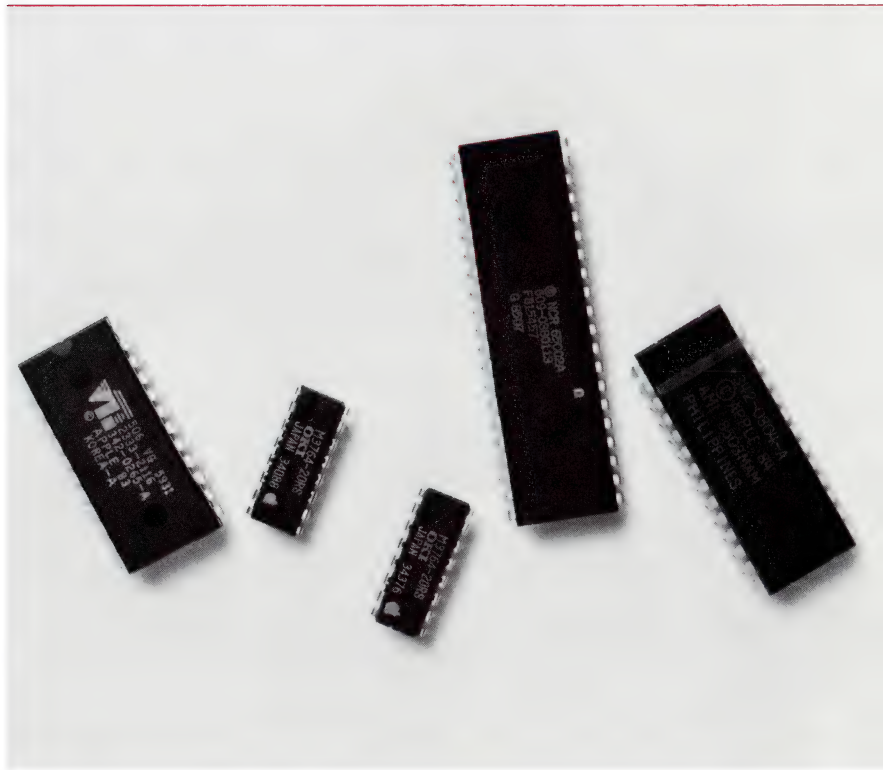
Note: † = New Address Modes

■ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

		IMME- DIATE	ABS- OLUTE	ZERO PAGE	ACCUM	IM- PLIED	(IND, X)	(IND, Y)	ZPG, X	ZPG, Y	ABS, X	ABS, Y	RELA- TIVE	(ABS)	ABS (IND, X)	(ZPG)	PROCESSOR STATUS CODES									
MNE	OPERATION	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	OP n	7	6	5	4	3	2	1	0	MNE	
ADC	A ← M + C ← A	(1,3)	88 2 2	8D 4 3	85 3 2			61 6 2	71 5 2	2 2	75 4 2	7D 4 3	78 4 3				72 5 2	N	V					Z	ADC	
AND	A ← A ∧ A	(1)	29 2 2	2D 4 3	25 3 2			21 6 2	31 5 2	2 2	35 4 2	3D 4 3	38 4 3				32 5 2	N						Z	AND	
ASL		(2)		0E 6 3	06 5 2	0A 2 1					16 6 2	1E 6 3												Z	ASL	
BCC	Branch if C=0	(2)												90 2 2										Z	BCC	
BCS	Branch if C=1	(2)												80 2 2										Z	BCS	
BEQ	Branch if Z=1	(2)												F0 2 2										Z	BEQ	
BIT	A ← A ∧ M	(4,5)	88 2 2	2C 4 3	24 3 2					34 4 2		3C 4 3												Z	BIT	
BMI	Branch if N=1	(2)												30 2 2										Z	BMI	
BNE	Branch if Z=0	(2)												D0 2 2										Z	BNE	
BPL	Branch if N=0	(2)												10 2 2										Z	BPL	
BRA	Branch Always	(2)												80 2 2										Z	BRA	
BRK	Break						00 7 1																	Z	BRK	
BVC	Branch if V=0	(2)												50 2 2										Z	BVC	
BVS	Branch if V=1	(2)												70 2 2										Z	BVS	
CLC	0 ← C							18 2 1																Z	CLC	
CLD	0 ← D							D8 2 1																Z	CLD	
CLI	0 ← I							58 2 1																Z	CLI	
CLV	0 ← V							88 2 1																Z	CLV	
CMP	A ← M	(1)	C9 2 2	CD 4 3	C5 3 2			C1 6 2	D1 5 2	D5 4 2		DD 4 3	D9 4 3				D2 5 2							Z	CMP	
CPX	X ← M		E0 2 2	EC 4 3	E4 3 2																			Z	CPX	
CPY	Y ← M																							Z	CPY	
DEA	A ← 1 ← A						3A 2 1																	Z	DEA	
DEC	M ←																									

Notes:

- | | | | |
|--|---|----------------|-----------------------------|
| 1. Add 1 to "n" if page boundary is crossed. | X Index X | + Add | n No. Cycles |
| 2. Add 1 to "n" if branch occurs to same page. | Y Index Y | - Subtract | # No. Bytes |
| 3. Add 2 to "n" if branch occurs to different page. | A Accumulator | & And | M ₆ Memory bit 6 |
| 4. Add 1 to "n" if decimal mode. | M Memory per effective address | V Or | M ₇ Memory bit 7 |
| 5. V bit equals memory bit 6 prior to execution. | M _s Memory per stack pointer | ⊕ Exclusive or | |
| 6. N bit equals memory bit 7 prior to execution. | | | |
| 7. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged. | | | |



Here is a list of useful subroutines in the Apple II's Monitor. To use these subroutines from machine-language programs, store data into the specified memory locations or microprocessor registers as required by the subroutine and execute a JSR to the subroutine's starting address. After the subroutine performs its function, it returns with the 65C02's registers changed as described.

▲Warning

For the sake of compatibility between the Apple II Plus, Apple IIc, and the Apple IIe, do not jump into the middle of Monitor subroutines. The starting addresses are the same for all models of the Apple II, but the actual code is different.

BASICIN Read the keyboard \$C305

When the 80-column firmware is active, BASICIN is used instead of KEYIN. BASICIN operates like KEYIN except that it displays a solid, non-blinking cursor instead of a blinking checkerboard cursor.

BASICOUT Output to screen \$C307

When the 80-column firmware is active, BASICOUT is used instead of COUT1. BASICOUT displays the character in the accumulator on the Apple II's screen at the current output cursor position and advances the output cursor. It places the character using the setting of the Normal/Inverse location. It handles control codes; see Table 3-3b. BASICOUT returns with all registers intact.

BELL Output a bell character \$FF3A

BELL writes a bell (Control-G) character to the current output device. It leaves the accumulator holding \$87.

BELL1 Sends a beep to the speaker \$FBDD

BELL1 generates a 1 kHz tone in the Apple II's speaker for 0.1 second. It scrambles the A and X registers.

CLREOL Clear to end of line \$FC9C

CLREOL clears a text line from the cursor position to the right edge of the window. CLREOL destroys the contents of A and Y.

CLEOLZ Clear to end of line \$FC9E

CLEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL, which is indexed by the contents of the Y register. CLEOLZ destroys the contents of A and Y.

CLREOP Clear to end of window \$FC42

CLREOP clears the text window from the cursor position to the bottom of the window. CLREOP destroys the contents of A and Y.

CLRSCR Clear the low-resolution screen \$F832

CLRSCR clears the low-resolution graphics display to black. If you call CLRSCR while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. CLRSCR destroys the contents of A and Y.

CLRTOP Clear the low-resolution screen \$F836

CLRTOP is the same as CLRSCR (above), except that it clears only the top 40 rows of the low-resolution display.

COUT Output a character \$FDED

COUT calls the current character output subroutine. The character to be output should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations \$36 and \$37), which is usually one of the standard character output subroutines, COUT1 or BASICOUT.

COUT1 Output to screen \$FDF0

COUT1 displays the character in the accumulator on the Apple IIe's screen at the current output cursor position and advances the output cursor. It places the character using the setting of the Normal/Inverse location. It handles the codes for carriage return, linefeed, backspace, and bell. It returns with all registers intact.

CROUT Generate a carriage return character \$FD8E

CROUT sends a carriage return character to the current output device.

CROUT1 Generate carriage return, clear rest of line \$FD8B

CROUT1 clears the screen from the current cursor position to the edge of the text window, then calls CROUT.

GETLN Get an input line with prompt \$FD6A

GETLN is the standard input subroutine for entire lines of characters, as described in Chapter 3. Your program calls GETLN with the prompt character in location \$33; GETLN returns with the input line in the input buffer (beginning at location \$0200) and the X register holding the length of the input line.

GETLNZ Get an input line \$FD67

GETLNZ is an alternate entry point for GETLN that sends a carriage return to the standard output, then continues into GETLN.

GETLN1 Get an input line, no prompt \$FD6F

GETLN1 is an alternate entry point for GETLN that does not issue a prompt before it accepts the input line. If, however, the user cancels the input line, either with too many backspaces or with a **CONTROL-X**, then GETLN1 will issue the contents of location \$33 as a prompt when it gets another line.

HLINE Draw a horizontal line of blocks \$F819

HLINE draws a horizontal line of blocks of the color set by SETCOL on the low-resolution graphics display. Call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLINE returns with A and Y scrambled, X intact.

HOME Home cursor and clear \$FC58

HOME clears the display and puts the cursor in the home position: the upper-left corner of the screen.

IOREST Restore all registers \$FF3F

IOREST loads the 65C02's internal registers with the contents of memory locations \$45 through \$49.

IOSAVE Save all registers \$FF4A

IOSAVE stores the contents of the 65C02's internal registers in locations \$45 through \$49 in the order A, X, Y, P, S. The contents of A and X are changed and the decimal mode is cleared.

KEYIN	Read the keyboard	\$FD1B
KEYIN is the keyboard input subroutine. It reads the Apple IIe's keyboard, waits for a keypress, and randomizes the random number seed at \$4E-\$4F. When a key is pressed, KEYIN removes the blinking cursor from the display and returns with the keycode in the accumulator. KEYIN is described in Chapter 3.		
MOVE	Move a block of memory	\$FE2C
MOVE copies the contents of memory from one range of locations to another. This subroutine is the same as the MOVE command in the Monitor, except that it takes its arguments from pairs of locations in memory, low-byte first. The destination address must be in A4 (\$42-\$43), the starting source address in A1 (\$3C-\$3D), and the ending source address in A2 (\$3E-\$3F) when your program calls MOVE. Register Y must contain \$00 when your program calls MOVE.		
NEXTCOL	Increment color by 3	\$F85F
NEXTCOL adds 3 to the current color (set by SETCOL) used for low-resolution graphics.		
PLOT	Plot on the low-resolution screen	\$F800
PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. The block's vertical position is passed in the accumulator, its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.		
PRBLNK	Print three spaces	\$F948
PRBLNK outputs three blank spaces to the standard output device. On return, the accumulator usually contains \$A0, the X register contains 0.		
PRBL2	Print many blank spaces	\$F94A
PRBL2 outputs from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to be output. If X=\$00, then PRBL2 will output 256 blanks.		
PRBYTE	Print a hexadecimal byte	\$FDDA
PRBYTE outputs the contents of the accumulator in hexadecimal on the current output device. The contents of the accumulator are scrambled.		

PREAD	Read a hand control	\$FB1E
PREAD returns a number that represents the position of a hand control. You pass the number of the hand control in the X register. If this number is not valid (not equal to 0, 1, 2, or 3), strange things may happen. PREAD returns with a number from \$00 to \$FF in the Y register. The accumulator is scrambled.		
PRERR	Print ERR	\$FF2D
PRERR sends the word <i>ERR</i> , followed by a bell character, to the standard output device. On return, the accumulator is scrambled.		
PRHEX	Print a hexadecimal digit	\$FDE3
PRHEX prints the lower nibble of the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.		
PRNTAX	Print A and X in hexadecimal	\$F941
PRNTAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte output, the X register contains the second. On return, the contents of the accumulator are scrambled.		
RDCHAR	Get an input character or escape code	\$FD35
RDCHAR is an alternate input subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.		
RDKEY	Get an input character	\$FD0C
RDKEY is the character input subroutine. It places a blinking cursor on the display at the cursor position and jumps to the subroutine whose address is stored in KSW (locations \$38 and \$39), usually the standard input subroutine KEYIN, which returns with a character in the accumulator.		
READ	Read a record from a cassette	\$FEFD
READ reads a series of tones at the cassette input port, converts them to data bytes, and stores the data in a specified range of memory locations. Before calling READ, the address of the first byte must be in A1 (\$3C-\$3D) and the address of the last byte must be in A2 (\$3E-\$3F).		

READ keeps a running exclusive-OR of the data bytes in CHKSUM (\$2E). When the last memory location has been filled, READ reads one more byte and compares it with CHKSUM. If they are equal, READ sends out a beep and returns; if not, it sends the string *ERR* through COUT, sends the beep, and returns.

SCRN Read the low-resolution graphics screen \$F871

SCRN returns the color value of a single block on the low-resolution graphics display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. Call it as you would call PLOT (above). The color of the block will be returned in the accumulator. No other registers are changed.

SETCOL Set low-resolution graphics color \$F864

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 2-6.

SETINV Set inverse mode \$FE80

SETINV sets the display format to inverse. COUT1 will then display all output characters as black dots on a white background. The Y register is set to \$3F, all others are unchanged.

SETNORM Set normal mode \$FE84

SETNORM sets the display format to normal. COUT1 will then display all output characters as white dots on a black background. On return, the Y register is set to \$FF, all others are unchanged.

VERIFY Compare two blocks of memory \$FE36

VERIFY compares the contents of one range of memory to another. This subroutine is the same as the VERIFY command in the Monitor, except it takes its arguments from pairs of locations in memory, low-byte first. The destination address must be in A4 (\$42-\$43), the starting source address in A1 (\$3C-\$3D), and the ending source address in A2 (\$3E-\$3F) when your program calls VERIFY.

VLINE Draw a vertical line of blocks \$F828

VLINE draws a vertical line of blocks of the color set by SETCOL on the low-resolution display. You should call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLINE will return with the accumulator scrambled.

WAIT Delay \$FCA8

WAIT delays for a specific amount of time, then returns to the program that called it. The amount of delay is specified by the contents of the accumulator. The delay is $1/2(26+27A+5A^2)$ microseconds, where A is the contents of the accumulator. WAIT returns with the accumulator zeroed and the X and Y registers undisturbed.

WRITE Write a record on a cassette \$FECD

WRITE converts the data in a range of memory to a series of tones at the cassette output port. Before calling WRITE, the address of the first data byte must be in A1 (\$3C-\$3D) and the address of the last byte must be in A2 (\$3E-\$3F). The subroutine writes a ten-second continuous tone as a header, then writes the data followed by a one-byte checksum.



This appendix lists the differences among the Apple II Plus, the original and the enhanced Apple IIe, and the Apple IIc.



If you're trying to write software to run on more than one version of the Apple II, this appendix will help you avoid unexpected problems of incompatibility.

The differences are listed here in approximately the order you are likely to encounter them: obvious differences first, technical details later. Each entry in the list includes references to the chapters in this manual where the item is described.

Keyboard

The Apple IIe and Apple IIc have a full 62-key uppercase and lowercase keyboard. The keyboard includes fully-operational **SHIFT** and **CAPS LOCK** keys. It also includes four directional arrow keys for moving the cursor. Chapter 2 includes a description of the keyboard. The cursor-motion keys are described in Chapter 3.

Apple Keys

The keyboard of the Apple IIe and Apple IIc have two keys marked with the Apple logo. These keys, called the Open-Apple key () and Solid-Apple key (), are used with the **RESET** key to select special reset functions. They are connected to the buttons on the hand controls, so they can be used for special functions in programs.

The Apple II and the Apple II Plus do not have Apple keys.

Character Sets

The Apple IIe and Apple IIc can display the full ASCII character set, uppercase and lowercase. For compatibility with older Apple II's, the standard display character set includes flashing uppercase instead of inverse-format lowercase; you can also switch to an alternate character set with inverse lowercase and uppercase, but no flashing. Chapter 2 includes a description of the display character sets. Chapter 3 tells you how to switch display formats.

The Apple IIc and the enhanced Apple IIe include a set of “graphic” text characters, called MouseText characters, that replace some of the inverse uppercase characters in the alternate character set of the original Apple IIe. MouseText characters are described in Chapter 2.

80-Column Display

With the addition of an 80-column text card, the Apple IIe can display 80 columns of text. The 80-column display is completely compatible with both graphics modes—you can even use it in mixed mode. (If you prefer, you can use an old-style 80-column card in an expansion slot instead.) Chapter 2 includes a description of the 80-column display.

The Apple IIc has a built-in extended 80-column card.

Escape Codes and Control Characters

On the Apple IIe and Apple IIc, the display features mentioned above (and many others not mentioned) can be controlled from the keyboard by escape sequences and from programs by control characters. Chapter 3 includes descriptions of those escape codes and control characters.

Built-in Language Card

The 16K bytes of RAM you add to the Apple II Plus by installing the Language Card is built into the Apple IIe and Apple IIc, giving the Apple IIe a standard memory size of 64K bytes. (The Apple IIc has a built-in extended 80-column text card as well, giving it a standard memory size of 128K bytes.) In the Apple IIe, this 16K-byte block of memory is called the bank-switched memory. It is described in Chapter 4.

Auxiliary Memory

By installing the Apple IIe Extended 80-Column Text Card, you can add an alternate 64K bytes of RAM to the Apple IIe. Chapter 4 tells you how to use the additional memory. (The Extended 80-Column Text Card also provides the 80-column display option.)

The Apple IIc has a built-in extended 80-column text card.

Auxiliary Slot

In addition to the expansion slots on the Apple II Plus, the Apple IIe has a special slot that is used either for the 80-Column Text Card or for the Extended 80-Column Text Card. This slot is identified in Chapter 1 and described in Chapter 7.

The Apple IIc has the functions of the auxiliary slot built in.

Back Panel and Connectors

The Apple IIe has a metal back panel with space for several D-type connectors. Each peripheral card you add comes with a connector that you install in the back panel. Chapter 1 includes a description of the back panel; for details, see the installation instructions supplied with the peripheral cards.

The Apple IIc back panel has seven built-in connectors.

Soft Switches

The display and memory features of the Apple IIe and the Apple IIc are controlled by soft switches like the ones on the Apple II Plus. On the Apple IIe and the Apple IIc, programs can also read the settings of the soft switches. Chapter 2 describes the soft switches that control the display features, and Chapter 4 describes the soft switches that control the memory features.

Built-in Self-Test

The Apple IIe has built-in firmware that includes a self-test routine. The self-test is intended primarily for testing during manufacturing, but you can run it to be sure the Apple IIe is working correctly. The self-test is described in Chapter 4.

The Apple IIc also has built-in diagnostics.

Forced Reset

Some programs on the Apple II Plus take control of the reset function to keep users from stopping the machine and copying the program. The Apple IIe and Apple IIc have a forced reset that writes over the program in memory. By using the forced reset, you can restart the Apple IIe (or Apple IIc) without turning power off and on and causing unnecessary stress on the circuits. The forced reset is described in Chapter 4.

Interrupt Handling

Even though most application programs don't use interrupts, the Apple IIe (and Apple IIc) provide for interrupt-driven programs. For example, the 80-column firmware periodically enables interrupts while it is clearing the display (normally a long time to have interrupts locked out). Interrupts are discussed in Chapter 6.

Vertical Sync for Animators

Programs with animation on the Apple IIe and Apple IIc can stay in step with the display and avoid flickering objects in their displays. Chapter 7 includes a description of the video generation and the vertical sync.

Signature Byte

A program can find out whether it's running on an Apple IIe, Apple IIc, Apple III (in emulation mode), or on an older model Apple II by reading the byte at location \$FBB3 in the System Monitor. In the Apple IIe Monitor, this byte's value is \$06; in the Autostart Monitor (the standard Monitor on the Apple II Plus), its value is \$EA. (Note: if you start up with DOS and switch to Integer BASIC, the Autostart Monitor is active and the value at location \$FBB3 is \$EA, even on an Apple IIe.) Obviously, there are lots of other locations that have different values in the different versions of the Monitor; location \$FBB3 was chosen because it will have the value \$06 even in future revisions of the Apple IIe Monitor.

Hardware Implementation

All of these features are described in Chapter 7.

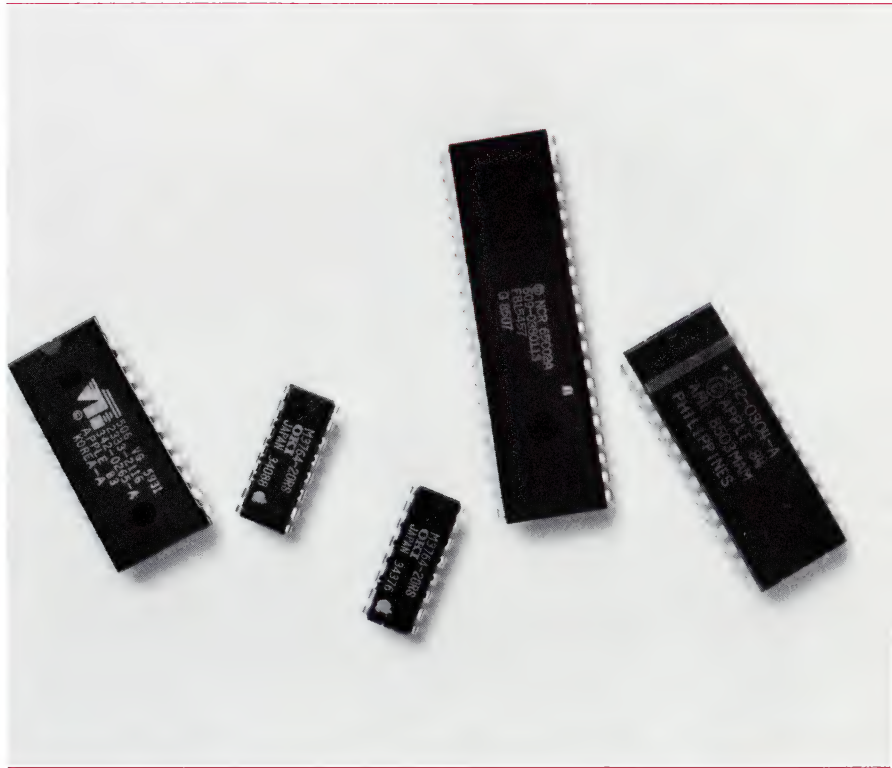
The hardware implementation of the Apple IIe is radically different from the Apple II and Apple II Plus. Three of the more important differences are

- the custom ICs: the IOU and MMU
- the video hardware, which uses ROM to generate both text and graphics
- the peripheral data bus, which is fully buffered.

The Apple IIc

For more information about the Apple IIc, see the *Apple IIc Reference Manual*.

- shares some of the custom ICs of the Apple IIe
- has some new ones all its own
- lacks the slots of the Apple IIe, replacing some of them with built-in I/O ports.



This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIe. It is not intended to be a full account. For more information, refer to the manuals that are provided with each product.

Operating Systems

This section discusses the operating systems that can be used with the Apple IIe.

ProDOS

ProDOS is the preferred disk operating system for the Apple IIe. It supports interrupts, startup from drives other than a Disk II, and all other hardware and firmware features of the Apple IIe.

DOS 3.3

The Apple IIe works with DOS 3.3. The Apple IIe can also access DOS 3.2 disks by using the *BASICS* disk. However, neither version of DOS takes full advantage of the features of the Apple IIe. DOS support is provided only for the sake of Apple II series compatibility.

Pascal Operating System

The Apple II Pascal operating system was developed from the UCSD Pascal system from the University of California at San Diego. While it shares many characteristics of that system, it has been extended by Apple in several areas.

Pascal versions 1.2 and later support interrupts and all the hardware and firmware features of the Apple IIe.

The Apple II Pascal system uses a disk format different than either ProDOS or DOS 3.3.

CP/M

CP/M® is an operating system developed by Digital Research that runs on either the Intel 8080 or Zilog Z80® microprocessors. This means that a co-processor peripheral card, available from several manufacturers for the Apple IIe, is required to run CP/M. Several versions of CP/M from 1.4 through 3.0 and later can be run on an Apple IIe with an appropriate co-processor card.

Languages

This section discusses special techniques to use, and characteristics to be aware of, when using Apple programming languages with the Apple IIe.

Assembly Language

An aid for assembly-language programming is *ProDOS Assembler Tools* (A2W0013).

Programs written in assembly language have the potential of extracting the most speed and efficiency from your Apple IIe, but they also require the most effort on your part.

Applesoft BASIC

The focus of the chapters in this manual is assembly language, and so most addresses and values are given in hexadecimal notation. Appendix E in this manual includes tables to help you convert from hexadecimal to the decimal notation you will need for BASIC.

In BASIC, use a PEEK to read a location (instead of the LDA used in assembly language), and a POKE (instead of STA) to write to a location. If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 holds a place value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

Integer BASIC

Integer BASIC is not included in the Apple IIe firmware. If you want to run it on your Apple IIe, you must use DOS 3.3 to load it in to the system. ProDOS does not support Integer BASIC.

Pascal Language

The Pascal language works on the Apple IIe under versions 1.1 and later of the Pascal Operating System. However, for best performance, use Pascal 1.2 or a later version.

FORTRAN

FORTRAN works under version 1.1 of the Pascal Operating System which does not detect or use certain Apple IIe features, such as auxiliary memory. Therefore, FORTRAN does not take advantage of these features.



This appendix briefly discusses bits and bytes and what they can represent. It also contains conversion tables for hexadecimal to decimal and negative decimal, for low-resolution display dot patterns, display color values, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

Bits and Bytes

This section discusses the relationships between bit values and their position within a byte. The following are some rules of thumb regarding the 65C02 and 6502.

- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIe are listed in Table E-1.

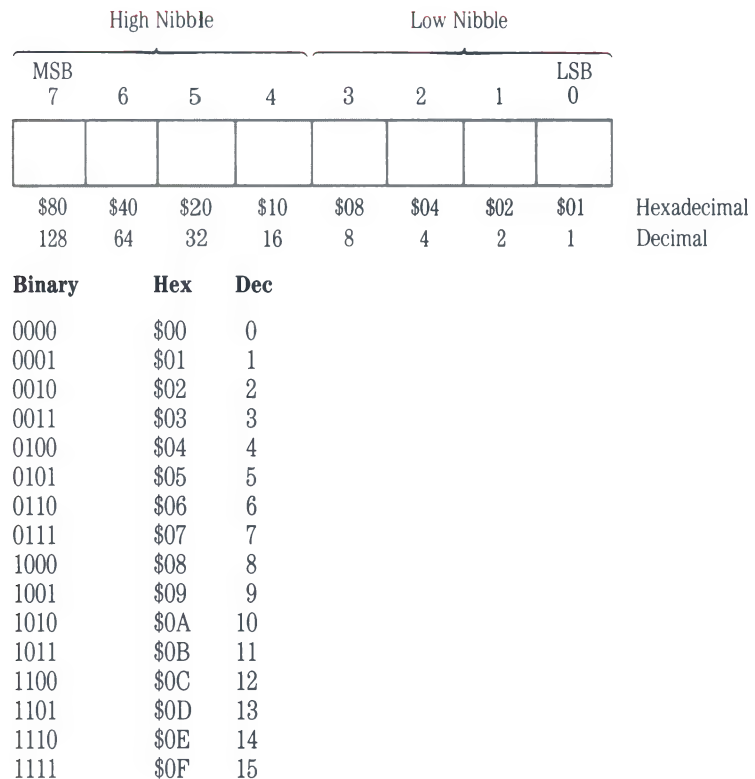
Table E-1. What a Bit Can Represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear *	Set
Serial transfer	Beginning	Start	Carrier (no informa tion yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

* Sometimes ambiguously termed *reset*.

- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only ten of the sixteen digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure E-1).

Figure E-1. Bits, Nibbles, and Bytes



- One byte can represent any of 16 x 16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.

- One memory position in the Apple IIe contains one eight-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data.
- Two bytes make a word. The sixteen bits of a word can represent any one of 256 x 256 or 65536 different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

Hexadecimal and Decimal

Use Table E-2 for conversion of hexadecimal and decimal numbers.

Table E-2. Hexadecimal/Decimal Conversion

Digit	\$x000	\$0x00	\$00x0	\$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

Examples:

\$3C = ?	\$FD47 = ?
\$30 = 48	\$F000 = 61440
\$0C = 12	\$D00 = 3328
-----	\$40 = 64
\$3C = 60	\$7 = 7

	\$FD47 = 64839

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than the number. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have zero left. Add up the hexadecimal numbers.

Example:

16215 = \$?	
16215 - 12288 = 3927	12288 = \$7000
3927 - 3840 = 87	3840 = \$F00
87 - 80 = 7	80 = \$50
7	7 = \$7

	16215 = \$7F57

Hexadecimal and Negative Decimal

If a number is larger than decimal 32767, Applesoft BASIC allows and Integer BASIC requires that you use the negative-decimal equivalent of the number. Table E-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative decimal number.

Table E-3. Hexadecimal to Negative Decimal Conversion

Digit	\$x000	\$\$0x00	\$\$\$00x0	\$\$\$\$000x
F	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8
7		-2048	-128	-9
6		-2304	-144	-10
5		-2560	-160	-11
4		-2816	-176	-12
3		-3072	-192	-13
2		-3328	-208	-14
1		-3584	-224	-15
0		-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (zeros included). Then add their values. The resulting number is the desired negative decimal number.

Example:

```
$C010 = - ?  
  
$C000:  -12288  
$ 000:  - 3840  
$  10:  -  224  
$   0:  -   16  
-----  
$C010  -16368
```

To convert a negative-decimal number to a positive decimal number, add it to 65536. (This addition ends up looking like subtraction.)

Example:

$$-151 = + ?$$

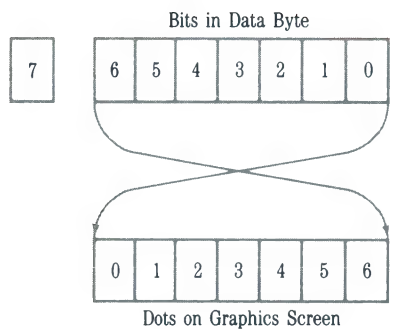
$$65536 + (-151) = 65536 - 151 = 65385$$

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive decimal number, then use Table E-2.

Graphics Bits and Pieces

Table E-4 is a quick guide to the hexadecimal values corresponding to 7-bit high-resolution patterns on the display screen. Since the bits are displayed in reverse order, it takes some calculation to determine these values. Table E-4 should make it easy.

Table E-4. Hexadecimal Values for High-Resolution Dot Patterns



Bit Pattern	x=0	x=1	Bit Pattern	x=0	x=1
x0000000	\$00	\$80	x0100000	\$02	\$82
x0000001	\$40	\$C0	x0100001	\$42	\$C2
x0000010	\$20	\$A0	x0100010	\$22	\$A2
x0000011	\$60	\$E0	x0100011	\$62	\$E2
x0000100	\$10	\$90	x0100100	\$12	\$92
x0000101	\$50	\$D0	x0100101	\$52	\$D2
x0000110	\$30	\$B0	x0100110	\$32	\$B2
x0000111	\$70	\$F0	x0100111	\$72	\$F2
x0001000	\$08	\$88	x0101000	\$0A	\$8A
x0001001	\$48	\$C8	x0101001	\$4A	\$CA
x0001010	\$28	\$A8	x0101010	\$2A	\$AA
x0001011	\$68	\$E8	x0101011	\$6A	\$EA
x0001100	\$18	\$98	x0101100	\$1A	\$9A
x0001101	\$58	\$D8	x0101101	\$5A	\$DA
x0001110	\$38	\$B8	x0101110	\$3A	\$BA
x0001111	\$78	\$F8	x0101111	\$7A	\$FA
x0010000	\$04	\$84	x0110000	\$06	\$86
x0010001	\$44	\$C4	x0110001	\$46	\$C6
x0010010	\$24	\$A4	x0110010	\$26	\$A6
x0010011	\$64	\$E4	x0110011	\$66	\$E6
x0010100	\$14	\$94	x0110100	\$16	\$96
x0010101	\$54	\$D4	x0110101	\$56	\$D6
x0010110	\$34	\$B4	x0110110	\$36	\$B6
x0010111	\$74	\$F4	x0110111	\$76	\$F6
x0011000	\$0C	\$8C	x0111000	\$0E	\$8E
x0011001	\$4C	\$CC	x0111001	\$4E	\$CE
x0011010	\$2C	\$AC	x0111010	\$2E	\$AE
x0011011	\$6C	\$EC	x0111011	\$6E	\$EE
x0011100	\$1C	\$9C	x0111100	\$1E	\$9E
x0011101	\$5C	\$DC	x0111101	\$5E	\$DE
x0011110	\$3C	\$BC	x0111110	\$3E	\$BE
x0011111	\$7C	\$FC	x0111111	\$7E	\$FE

The x represents bit 7. Zeros represent bits that are off; ones bits that are on. Use the first hexadecimal value if bit 7 is to be off, and the second if it is to be on.

For example, to get bit pattern 00101110, use \$3A; for 10101110, use \$BA.

Table E-4—Continued. Hexadecimal Values for High-Resolution Dot Patterns

Bit Pattern	x=0	x=1	Bit Pattern	x=0	x=1
x1000000	\$01	\$81	x1100000	\$03	\$83
x1000001	\$41	\$C1	x1100001	\$43	\$C3
x1000010	\$21	\$A1	x1100010	\$23	\$A3
x1000011	\$61	\$E1	x1100011	\$63	\$E3
x1000100	\$11	\$91	x1100100	\$13	\$93
x1000101	\$51	\$D1	x1100101	\$53	\$D3
x1000110	\$31	\$B1	x1100110	\$33	\$B3
x1000111	\$71	\$F1	x1100111	\$73	\$F3
x1001000	\$09	\$89	x1101000	\$0B	\$8B
x1001001	\$49	\$C9	x1101001	\$4B	\$CB
x1001010	\$29	\$A9	x1101010	\$2B	\$AB
x1001011	\$69	\$E9	x1101011	\$6B	\$EB
x1001100	\$19	\$99	x1101100	\$1B	\$9B
x1001101	\$59	\$D9	x1101101	\$5B	\$DB
x1001110	\$39	\$B9	x1101110	\$3B	\$BB
x1001111	\$79	\$F9	x1101111	\$7B	\$FB
x1010000	\$05	\$85	x1110000	\$07	\$87
x1010001	\$45	\$C5	x1110001	\$47	\$C7
x1010010	\$25	\$A5	x1110010	\$27	\$A7
x1010011	\$65	\$E5	x1110011	\$67	\$E7
x1010100	\$15	\$95	x1110100	\$17	\$97
x1010101	\$55	\$D5	x1110101	\$57	\$D7
x1010110	\$35	\$B5	x1110110	\$37	\$B7
x1010111	\$75	\$F5	x1110111	\$77	\$F7
x1011000	\$0D	\$8D	x1111000	\$0F	\$8F
x1011001	\$4D	\$CD	x1111001	\$4F	\$CF
x1011010	\$2D	\$AD	x1111010	\$2F	\$AF
x1011011	\$6D	\$ED	x1111011	\$6F	\$EF
x1011100	\$1D	\$9D	x1111100	\$1F	\$9F
x1011101	\$5D	\$DD	x1111101	\$5F	\$DF
x1011110	\$3D	\$BD	x1111110	\$3F	\$BF
x1011111	\$7D	\$FD	x1111111	\$7F	\$FF

Eight-Bit Code Conversions

Tables E-5 through E-12 show the entire ASCII character set twice: once with the high bit off, and once with it on. Here is how to interpret these tables.

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 010010000 for the letter *H*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H*.
- A transmitted or received ASCII character will take whichever form is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity H, 01001000 for an even-parity H.
- The *ASCII Char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to Type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Pri* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

Note that the values \$40 through \$5F (and \$C0 through \$DF) in the alternate character set are displayed as MouseText characters if MouseText is turned on.

The MouseText characters are shown in Table E-7.

Note: The primary and alternate displayed character sets in Tables E-5 through E-12 are the result of firmware mapping. The character generator ROM actually contains only one character set. The firmware mapping procedure is described in the section “Inverse and Flashing Text,” in Chapter 3.

Table E-5. Control Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0000000	0	\$00	NUL	Blank (null)	CONTROL-@	@	@
0000001	1	\$01	SOH	Start of Header	CONTROL-A	A	A
0000010	2	\$02	STX	Start of Text	CONTROL-B	B	B
0000011	3	\$03	ETX	End of Text	CONTROL-C	C	C
0000100	4	\$04	EOT	End of Transm.	CONTROL-D	D	D
0000101	5	\$05	ENQ	Enquiry	CONTROL-E	E	E
0000110	6	\$06	ACK	Acknowledge	CONTROL-F	F	F
0000111	7	\$07	BEL	Bell	CONTROL-G	G	G
0001000	8	\$08	BS	Backspace	CONTROL-H or ←	H	H
0001001	9	\$09	HT	Horizontal Tab	CONTROL-I or TAB	I	I
0001010	10	\$0A	LF	Line Feed	CONTROL-J or ↓	J	J
0001011	11	\$0B	VT	Vertical Tab	CONTROL-K or ↑	K	K
0001100	12	\$0C	FF	Form Feed	CONTROL-L	L	L
0001101	13	\$0D	CR	Carriage Return	CONTROL-M or RETURN	M	M
0001110	14	\$0E	SO	Shift Out	CONTROL-N	N	N
0001111	15	\$0F	SI	Shift In	CONTROL-O	O	O
0010000	16	\$10	DLE	Data Link Escape	CONTROL-P	P	P
0010001	17	\$11	DC1	Device Control 1	CONTROL-Q	Q	Q
0010010	18	\$12	DC2	Device Control 2	CONTROL-R	R	R
0010011	19	\$13	DC3	Device Control 3	CONTROL-S	S	S
0010100	20	\$14	DC4	Device Control 4	CONTROL-T	T	T
0010101	21	\$15	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
0010110	22	\$16	SYN	Synchronization	CONTROL-V	V	V
0010111	23	\$17	ETB	End of Text Blk.	CONTROL-W	W	W
0011000	24	\$18	CAN	Cancel	CONTROL-X	X	X
0011001	25	\$19	EM	End of Medium	CONTROL-Y	Y	Y
0011010	26	\$1A	SUB	Substitute	CONTROL-Z	Z	Z
0011011	27	\$1B	ESC	Escape	CONTROL-[or ESC	[[
0011100	28	\$1C	FS	File Separator	CONTROL-\	\	\
0011101	29	\$1D	GS	Group Separator	CONTROL-]]]
0011110	30	\$1E	RS	Record Separator	CONTROL-^	^	^
0011111	31	\$1F	US	Unit Separator	CONTROL-~	~	~

Table E-6. Special Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0100000	32	\$20	SP	Space	SPACE bar		
0100001	33	\$21	!			!	!
0100010	34	\$22	"			"	"
0100011	35	\$23	#			#	#
0100100	36	\$24	\$			\$	\$
0100101	37	\$25	%			%	%
0100110	38	\$26	&			&	&
0100111	39	\$27	'	Closing Quote		'	'
0101000	40	\$28	(((
0101001	41	\$29)))
0101010	42	\$2A	*			*	*
0101011	43	\$2B	+			+	+
0101100	44	\$2C	,	Comma		,	,
0101101	45	\$2D	-	Hyphen		-	-
0101110	46	\$2E	.	Period		.	.
0101111	47	\$2F	/			/	/
0110000	48	\$30	0			0	0
0110001	49	\$31	1			1	1
0110010	50	\$32	2			2	2
0110011	51	\$33	3			3	3
0110100	52	\$34	4			4	4
0110101	53	\$35	5			5	5
0110110	54	\$36	6			6	6
0110111	55	\$37	7			7	7
0111000	56	\$38	8			8	8
0111001	57	\$39	9			9	9
0111010	58	\$3A	:			:	:
0111011	59	\$3B	;			;	;
0111100	60	\$3C	<			<	<
0111101	61	\$3D	=			=	=
0111110	62	\$3E	>			>	>
0111111	63	\$3F	?			?	?

Table E-7. Uppercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1000000	64	\$40	@			@	Ⓐ
1000001	65	\$41	A			A	Ⓐ
1000010	66	\$42	B			B	Ⓑ
1000011	67	\$43	C			C	Ⓒ
1000100	68	\$44	D			D	Ⓓ
1000101	69	\$45	E			E	Ⓔ
1000110	70	\$46	F			F	Ⓕ
1000111	71	\$47	G			G	Ⓖ
1001000	72	\$48	H			H	Ⓗ
1001001	73	\$49	I			I	Ⓐ
1001010	74	\$4A	J			J	Ⓙ
1001011	75	\$4B	K			K	Ⓚ
1001100	76	\$4C	L			L	Ⓛ
1001101	77	\$4D	M			M	Ⓜ
1001110	78	\$4E	N			N	Ⓝ
1001111	79	\$4F	O			O	Ⓞ
1010000	80	\$50	P			P	Ⓟ
1010001	81	\$51	Q			Q	Ⓠ
1010010	82	\$52	R			R	Ⓡ
1010011	83	\$53	S			S	Ⓢ
1010100	84	\$54	T			T	Ⓣ
1010101	85	\$55	U			U	Ⓤ
1010110	86	\$56	V			V	Ⓥ
1010111	87	\$57	W			W	Ⓦ
1011000	88	\$58	X			X	Ⓧ
1011001	89	\$59	Y			Y	Ⓨ
1011010	90	\$5A	Z			Z	Ⓩ
1011011	91	\$5B	[Opening Bracket		[Ⓛ
1011100	92	\$5C	\	Reverse Slant		\	Ⓜ
1011101	93	\$5D]	Closing Bracket]	Ⓨ
1011110	94	\$5E	^	Caret		^	Ⓩ
1011111	95	\$5F	_	Underline		_	Ⓛ

Table E-8. Lowercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1100000	96	\$60	`	Opening Quote			`
1100001	97	\$61	a			!	a
1100010	98	\$62	b			"	b
1100011	99	\$63	c			#	c
1100100	100	\$64	d			\$	d
1100101	101	\$65	e			%	e
1100110	102	\$66	f			&	f
1100111	103	\$67	g			'	g
1101000	104	\$68	h			(h
1101001	105	\$69	i)	i
1101010	106	\$6A	j			*	j
1101011	107	\$6B	k			+	k
1101100	108	\$6C	l			,	l
1101101	109	\$6D	m			-	m
1101110	110	\$6E	n			.	n
1101111	111	\$6F	o			/	o
1110000	112	\$70	p			0	p
1110001	113	\$71	q			1	q
1110010	114	\$72	r			2	r
1110011	115	\$73	s			3	s
1110100	116	\$74	t			4	t
1110101	117	\$75	u			5	u
1110110	118	\$76	v			6	v
1110111	119	\$77	w			7	w
1111000	120	\$78	x			8	x
1111001	121	\$79	y			9	y
1111010	122	\$7A	z			:	z
1111011	123	\$7B	{	Opening Brace		;	{
1111100	124	\$7C		Vertical Line		<	
1111101	125	\$7D	}	Closing Brace		=	}
1111110	126	\$7E	~	Overline (Tilde)		>	~
1111111	127	\$7F	DEL	Delete/Rubout		?	DEL

Table E-9. Control Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10000000	128	\$80	NUL	Blank (null)	CONTROL-@	@	@
10000001	129	\$81	SOH	Start of Header	CONTROL-A	A	A
10000010	130	\$82	STX	Start of Text	CONTROL-B	B	B
10000011	131	\$83	ETX	End of Text	CONTROL-C	C	C
10000100	132	\$84	EOT	End of Transm.	CONTROL-D	D	D
10000101	133	\$85	ENQ	Enquiry	CONTROL-E	E	E
10000110	134	\$86	ACK	Acknowledge	CONTROL-F	F	F
10000111	135	\$87	BEL	Bell	CONTROL-G	G	G
10001000	136	\$88	BS	Backspace	CONTROL-H or ←	H	H
10001001	137	\$89	HT	Horizontal Tab	CONTROL-I or TAB	I	I
10001010	138	\$8A	LF	Line Feed	CONTROL-J or ↓	J	J
10001011	139	\$8B	VT	Vertical Tab	CONTROL-K or ↑	K	K
10001100	140	\$8C	FF	Form Feed	CONTROL-L	L	L
10001101	141	\$8D	CR	Carriage Return	CONTROL-M or RETURN	M	M
10001110	142	\$8E	SO	Shift Out	CONTROL-N	N	N
10001111	143	\$8F	SI	Shift In	CONTROL-O	O	O
10010000	144	\$90	DLE	Data Link Escape	CONTROL-P	P	P
10010001	145	\$91	DC1	Device Control 1	CONTROL-Q	Q	Q
10010010	146	\$92	DC2	Device Control 2	CONTROL-R	R	R
10010011	147	\$93	DC3	Device Control 3	CONTROL-S	S	S
10010100	148	\$94	DC4	Device Control 4	CONTROL-T	T	T
10010101	149	\$95	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
10010110	150	\$96	SYN	Synchronization	CONTROL-V	V	V
10010111	151	\$97	ETB	End of Text Blk.	CONTROL-W	W	W
10011000	152	\$98	CAN	Cancel	CONTROL-X	X	X
10011001	153	\$99	EM	End of Medium	CONTROL-Y	Y	Y
10011010	154	\$9A	SUB	Substitute	CONTROL-Z	Z	Z
10011011	155	\$9B	ESC	Escape	CONTROL-[or ESC	[[
10011100	156	\$9C	FS	File Separator	CONTROL-\	\	\
10011101	157	\$9D	GS	Group Separator	CONTROL-]]]
10011110	158	\$9E	RS	Record Separator	CONTROL-^	^	^
10011111	159	\$9F	US	Unit Separator	CONTROL-—	—	—

Table E-10. Special Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10100000	160	\$A0	SP	Space	SPACE bar		
10100001	161	\$A1	!			!	!
10100010	162	\$A2	"			"	"
10100011	163	\$A3	#			#	#
10100100	164	\$A4	\$			\$	\$
10100101	165	\$A5	%			%	%
10100110	166	\$A6	&			&	&
10100111	167	\$A7	'	Closed Quote (acute accent)		'	'
10101000	168	\$A8	(((
10101001	169	\$A9)))
10101010	170	\$AA	*			*	*
10101011	171	\$AB	+			+	+
10101100	172	\$AC	,	Comma		,	,
10101101	173	\$AD	-	Hyphen		-	-
10101110	174	\$AE	.	Period		.	.
10101111	175	\$AF	/			/	/
10110000	176	\$B0	0			0	0
10110001	177	\$B1	1			1	1
10110010	178	\$B2	2			2	2
10110011	179	\$B3	3			3	3
10110100	180	\$B4	4			4	4
10110101	181	\$B5	5			5	5
10110110	182	\$B6	6			6	6
10110111	183	\$B7	7			7	7
10111000	184	\$B8	8			8	8
10111001	185	\$B9	9			9	9
10111010	186	\$BA	:			:	:
10111011	187	\$BB	;			;	;
10111100	188	\$BC	<			<	<
10111101	189	\$BD	=			=	=
10111110	190	\$BE	>			>	>
10111111	191	\$BF	?			?	?

Table E-11. Uppercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11000000	192	\$C0	@			@	@
11000001	193	\$C1	A			A	A
11000010	194	\$C2	B			B	B
11000011	195	\$C3	C			C	C
11000100	196	\$C4	D			D	D
11000101	197	\$C5	E			E	E
11000110	198	\$C6	F			F	F
11000111	199	\$C7	G			G	G
11001000	200	\$C8	H			H	H
11001001	201	\$C9	I			I	I
11001010	202	\$CA	J			J	J
11001011	203	\$CB	K			K	K
11001100	204	\$CC	L			L	L
11001101	205	\$CD	M			M	M
11001110	206	\$CE	N			N	N
11001111	207	\$CF	O			O	O
11010000	208	\$D0	P			P	P
11010001	209	\$D1	Q			Q	Q
11010010	210	\$D2	R			R	R
11010011	211	\$D3	S			S	S
11010100	212	\$D4	T			T	T
11010101	213	\$D5	U			U	U
11010110	214	\$D6	V			V	V
11010111	215	\$D7	W			W	W
11011000	216	\$D8	X			X	X
11011001	217	\$D9	Y			Y	Y
11011010	218	\$DA	Z			Z	Z
11011011	219	\$DB	[Opening Bracket		[[
11011100	220	\$DC	\	Reverse Slant		\	\
11011101	221	\$DD]	Closing Bracket]]
11011110	222	\$DE	^	Caret		^	^
11011111	223	\$DF	—	Underline		—	—

Table E-12. Lowercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11100000	224	\$E0	`	Open Quote		`	`
11100001	225	\$E1	a			a	a
11100010	226	\$E2	b			b	b
11100011	227	\$E3	c			c	c
11100100	228	\$E4	d			d	d
11100101	229	\$E5	e			e	e
11100110	230	\$E6	f			f	f
11100111	231	\$E7	g			g	g
11101000	232	\$E8	h			h	h
11101001	233	\$E9	i			i	i
11101010	234	\$EA	j			j	j
11101011	235	\$EB	k			k	k
11101100	236	\$EC	l			l	l
11101101	237	\$ED	m			m	m
11101110	238	\$EE	n			n	n
11101111	239	\$EF	o			o	o
11110000	240	\$F0	p			p	p
11110001	241	\$F1	q			q	q
11110010	242	\$F2	r			r	r
11110011	243	\$F3	s			s	s
11110100	244	\$F4	t			t	t
11110101	245	\$F5	u			u	u
11110110	246	\$F6	v			v	v
11110111	247	\$F7	w			w	w
11111000	248	\$F8	x			x	x
11111001	249	\$F9	y			y	y
11111010	250	\$FA	z			z	z
11111011	251	\$FB	{	Opening Brace		{	{
11111100	252	\$FC		Vertical Line			
11111101	253	\$FD	}	Closing Brace		}	}
11111110	254	\$FE	~	Overline (Tilde)		~	~
11111111	255	\$FF	DEL	Delete (Rubout)	DELETE	DEL	DEL



This appendix contains copies of the tables you will need to refer to frequently, for example, ASCII codes and soft-switch location. The figures all have their original figure numbers.

Table 2-3. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
←	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
↓	0A	LF	0A	LF	0A	LF	0A	LF
↑	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
→	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
' "	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
- _	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29)	29)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (39	9	39	9	28	(28	(
; :	3B	;	3B	;	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[{	5B	[1B	ESC	7B	{	1B	ESC
\	5C	\	1C	FS	7C		1C	FS
] } ~	5D]	1D	GS	7D	}	1D	GS
` ~	60	`	60	`	7E	~	7E	~

Table 2-3—Continued. Keys and ASCII Codes

Note: Codes are shown here in hexadecimal; to find the decimal equivalents, refer to Table E-2.

Key	Normal		Control		Shift		Both	
	Code	Char	Code	Char	Code	Char	Code	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

Table 2-2. Keyboard Memory Locations

Hex	Location		Description
	Decimal		
\$C000	49152	-16384	Keyboard data and strobe
\$C010	49168	-16368	Any-key-down flag and clear-strobe switch

Table 2-4. Video Display Specifications

Display modes:	40-column text; map: Figure 2-2 80-column text; map: Figure 2-3 Low-resolution color graphics; map: Figure 2-7 High-resolution color graphics; map: Figure 2-8 Double-high-resolution color graphics; map: Figure 2-9
Text capacity:	24 lines by 80 columns (character positions)
Character set:	96 ASCII characters (uppercase and lowercase)
Display formats:	Normal, inverse, flashing, MouseText (Table 2-5)
Low-resolution graphics:	16 colors (Table 2-6) 40 horizontal by 48 vertical; map: Figure 2-7
High-resolution graphics:	6 colors (Table 2-7) 140 horizontal by 192 vertical (restricted) Black-and-white: 280 horizontal by 192 vertical; map: Figure 2-8
Double-high-resolution graphics:	16 colors (Table 2-8) 140 horizontal by 192 vertical (no restrictions) Black-and-white: 560 horizontal by 192 vertical; map: Figure 2-9

Table 2-8. Double-High-Resolution Graphics Colors

Color	ab0	mb1	ab2	mb3	Repeated Bit Pattern
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark Green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$55	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark Blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium Blue	\$33	\$66	\$4C	\$19	1100
Light Blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

Table 2-9. Video Display Page Locations

Display Mode	Display Page	Lowest Address		Highest Address	
		Hex	Dec	Hex	Dec
40-column text, low-resolution graphics	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
80-column text	1	\$0400	1024	\$07FF	2047
	2 *	\$0800	2048	\$0BFF	3071
High-resolution graphics	1	\$2000	8192	\$3FFF	16383
	2	\$4000	16384	\$5FFF	24575
Double-high- resloution graphics	1 †	\$2000	8192	\$3FFF	16383
	2 †	\$4000	16384	\$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to the section “Display Mode Switching” in Chapter 2.

† See the section “Double-High-Resolution Graphics,” in Chapter 2.

Table 2-10. Display Soft Switches

Note: W means write anything to the location, R means read the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
ALTCHAR	W	\$C00E	Off: display text using primary character set
ALTCHAR	W	\$C00F	On: display text using alternate character set
RDALTCHAR	R7	\$C01E	Read ALTCHAR switch (1 = on)
80COL	W	\$C00C	Off: display 40 columns
80COL	W	\$C00D	On: display 80 columns
RD80COL	R7	\$C01F	Read 80COL switch (1 = on)
80STORE	W	\$C000	Off: cause PAGE2 on to select auxiliary RAM
80STORE	W	\$C001	On: allow PAGE2 to switch main RAM areas
RD80STORE	R7	\$C018	Read 80STORE switch (1 = on)
PAGE2	R/W	\$C054	Off: select Page 1
PAGE2	R/W	\$C055	On: select Page 2 or, if 80STORE on, Page 1 in auxiliary memory
RDPAGE2	R7	\$C01C	Read PAGE2 switch (1 = on)
TEXT	R/W	\$C050	Off: display graphics or, if MIXED on, mixed
TEXT	R/W	\$C051	On: display text
RDTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C052	Off: display only text or only graphics
MIXED	R/W	\$C053	On: if TEXT off, display text and graphics
RDMIXED	R7	\$C01B	Read MIXED switch (1 = on)
HIRES	R/W	\$C056	Off: if TEXT off, display low-resolution graphics
HIRES	R/W	\$C059	On: if TEXT off, display high-resolution or, if DHIRES on, double-high-resolution graphics
RDHIRES	R7	\$C01D	Read HIRES switch (1 = on)
IOUDIS	W	\$C07E	On: disable IOU access for addresses \$C058 to \$C05F; enable access to DHIRES switch *
IOUDIS	W	\$C07F	Off: enable IOU access for addresses \$C058 to \$C05F; disable access to DHIRES switch *
RДИОUDIS	R7	\$C07E	Read IOUDIS switch (1 = off) †
DHIRES	R/W	\$C05E	On: (if IOUDIS on) turn on double-high-res.
DHIRES	R/W	\$C05F	Off: (if IOUDIS on) turn off double-high-res.
RDDHIRES	R7	\$C07F	Read DHIRES switch (1 = on) †

* The firmware normally leaves IOUDIS on. See also †.

† Reading or writing any address in the range \$C070-\$C07F also triggers the paddle timer and resets VBLINT (Chapter 7).

Table 3-1. Monitor Firmware Routines

Location	Name	Description
\$C305	BASICIN	With 80-column firmware active, displays solid, blinking cursor. Accepts character from keyboard.
\$C307	BASICOUT	Displays a character on the screen; used when the 80-column firmware is active (Chapter 3).
\$FC9C	CLREOL	Clears to end of line from current cursor position.
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position.
\$FC42	CLREOP	Clears to bottom of window.
\$F832	CLRSCR	Clears the low-resolution screen.
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen.
\$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3).
\$FDF0	COUT1	Displays a character on the screen (Chapter 3).
\$FD8E	CROUT	Generates a carriage return character.
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character.
\$FD6A	GETLN	Displays the prompt character; accepts a string of characters by means of RDKEY.
\$F819	HLINE	Draws a horizontal line of blocks.
\$FC58	HOME	Clears window; puts cursor in upper-left corner of window.
\$FD1B	KEYIN	With 80-column firmware inactive, displays checkerboard cursor. Accepts character from keyboard.
\$F800	PLOT	Plots a single low-resolution block on the screen.
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device.
\$FDDA	PRBYTE	Prints a hexadecimal byte.
\$FF2D	PRERR	Sends ERR and Control-G to the output device.
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number.
\$F941	PRNTAX	Prints contents of A and X in hexadecimal.
\$FD0C	RDKEY	Displays blinking cursor; goes to standard input routine, normally KEYIN or BASICIN.
\$F871	SCRN	Reads color value of a low-resolution block.
\$F864	SETCOL	Sets the color for plotting in low-resolution.
\$FC24	VTABZ	Sets cursor vertical position.
\$F828	VLINE	Draws a vertical line of low-resolution blocks.

Table 3-3a. Control Characters With 80-Column Firmware Off

Control Character	ASCII Name	Apple IIe Name	Action Taken by COUT1
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.

Table 3-3b. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-K †	VT	clear EOS	Clears from cursor position to the end of the screen.
Control-L †	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.
Control-N †	SO	normal	Sets display format normal.
Control-O †	SI	inverse	Sets display format inverse.
Control-Q †	DC1	40-column	Sets display to 40-column.
Control-R †	DC2	80-column	Sets display to 80-column.
Control-S *	DC3	stop-list	Stops listing characters on the display until another key is pressed.

Table 3-3b—Continued. Control Characters With 80-Column Firmware On

Control Character	ASCII Name	Apple IIe Name	Action Taken by BASICOUT
Control-U †	NAK	quit	Deactivates 80-column video firmware.
Control-V †	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.
Control-W †	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.
Control-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.
Control-Y †	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).
Control-Z †	SUB	clear line	Clears the line the cursor position is on.
Control-[ESC	enable MouseText	Map inverse uppercase characters to MouseText characters.
Control-\ †	FS	forward space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.
Control-] †	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window).
Control-__	US	up	Moves cursor up a line, no scroll.

* Only works from the keyboard.

† Doesn't work from the keyboard.

Table 3-5. Text Format Control Values

Note: These mask values apply only to the primary character set (see text).

Mask Value		Display Format
Dec	Hex	
255	\$FF	Normal, uppercase, and lowercase
127	\$7F	Flashing, uppercase, and symbols
63	\$3F	Inverse, uppercase, and lowercase

Table 3-6. Escape Codes

Escape Code	Function
<code>ESC</code> <code>@</code>	Clears window and homes cursor (places it in upper-left corner of screen), then exits from escape mode.
<code>ESC</code> <code>A</code> or <code>a</code>	Moves cursor right one line; exits from escape mode.
<code>ESC</code> <code>B</code> or <code>b</code>	Moves cursor left one line; exits from escape mode.
<code>ESC</code> <code>C</code> or <code>c</code>	Moves cursor down one line; exits from escape mode.
<code>ESC</code> <code>D</code> or <code>d</code>	Moves cursor up one line; exits from escape mode.
<code>ESC</code> <code>E</code> or <code>e</code>	Clears to end of line; exits from escape mode.
<code>ESC</code> <code>F</code> or <code>f</code>	Clears to bottom of window; exits from escape mode.
<code>ESC</code> <code>I</code> or <code>i</code> or <code>ESC</code> <code>↑</code>	Moves the cursor up one line; remains in escape mode. See text.
<code>ESC</code> <code>J</code> or <code>j</code> or <code>ESC</code> <code>←</code>	Moves the cursor left one space; remains in escape mode. See text.
<code>ESC</code> <code>K</code> or <code>k</code> or <code>ESC</code> <code>→</code>	Moves the cursor right one space; remains in escape mode. See text.
<code>ESC</code> <code>M</code> or <code>m</code> or <code>ESC</code> <code>↓</code>	Moves the cursor down one line; remains in escape mode. See text.
<code>ESC</code> <code>4</code>	If 80-column firmware is active, switches to 40-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
<code>ESC</code> <code>8</code>	If 80-column firmware is active, switches to 80-column mode; sets links to BASICIN and BASICOUT; restores normal window size; exits from escape mode.
<code>ESC</code> <code>CONTROL</code> <code>D</code>	Disables control characters; only carriage return, line feed, BELL, and backspace have an effect when printed.
<code>ESC</code> <code>CONTROL</code> <code>E</code>	Reactivates control characters.
<code>ESC</code> <code>CONTROL</code> <code>Q</code>	If 80-column firmware is active, deactivates 80-column firmware; sets links to KEYIN and COUT1; restores normal window size; exits from escape mode.

Table 3-10. Pascal Video Control Functions

Control-	Hex	Function performed
E or e	\$05	Turns cursor on (enables cursor display).
F or f	\$06	Turns cursor off (disables cursor display).
G or g	\$07	Sounds bell (beeps).
H or h	\$08	Moves cursor left one column. If cursor was at beginning of line, moves it to end of previous line.
J or j	\$0A	Moves cursor down one row; scrolls if needed.
K or k	\$0B	Clears to end of screen.
L or l	\$0C	Clears screen; moves cursor to upper-left of screen.
M or m	\$0D	Moves cursor to column 0.
N or n	\$0E	Displays subsequent characters in normal video. (Characters already on display are unaffected.)
O or o	\$0F	Displays subsequent characters in inverse video. (Characters already on display are unaffected.)
V or v	\$16	Scrolls screen up one line; clears bottom line.
W or w	\$17	Scrolls screen down one line; clears top line.
Y or y	\$19	Moves cursor to upper-left (home) position on screen.
Z or z	\$1A	Clears entire line that cursor is on.
or \	\$1C	Moves cursor right one column; if at end of line, does Control-M.
} or]	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor.
^ or 6	\$1E	GOTOxy: initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively.
—	\$1F	If not at top of screen, moves cursor up one line.

Table 4-6. Bank Select Switches

Note: R means read the location, W means write anything to the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
	R	\$C080	Read RAM; no write; use \$D000 bank 2.
	RR	\$C081	Read ROM; write RAM; use \$D000 bank 2.
	R	\$C082	Read ROM; no write; use \$D000 bank 2.
	RR	\$C083	Read and write RAM; use \$D000 bank 2.
	R	\$C088	Read RAM; no write; use \$D000 bank 1.
	RR	\$C089	Read ROM; write RAM; use \$D000 bank 1.
	R	\$C08A	Read ROM; no write; use \$D000 bank 1.
	RR	\$C08B	Read and write RAM; use \$D000 bank 1.
RDBNK2	R7	\$C011	Read whether \$D000 bank 2 (1) or bank 1 (0)
RDLGRAM	R7	\$C012	Reading RAM (1) or ROM (0).
ALTZP	W	\$C008	Off: use main bank, page 0 and page 1.
ALTZP	W	\$C009	On: use auxiliary bank, page 0 and page 1.
RDALTZP	R7	\$C016	Read whether auxiliary (1) or main (0) bank

Table 4-7. Auxiliary-Memory Select Switches

Name	Function	Hex	Location		Notes
			Decimal		
RAMRD	Read auxiliary memory	\$C003	49155	-16381	Write
	Read main memory	\$C002	49154	-16382	Write
	Read RAMRD switch	\$C013	49171	-16365	Read
RAMWRT	Write auxiliary memory	\$C005	49157	-16379	Write
	Write main memory	\$C004	49156	-16380	Write
	Read RAMWRT switch	\$C014	49172	-16354	Read
80STORE	On: access display page	\$C001	49153	-16383	Write
	Off: use RAMRD, RAMWRT	\$C000	49152	-16384	Write
	Read 80STORE switch	\$C018	49176	-16360	Read
PAGE2	Page 2 on (aux. memory)	\$C055	49237	-16299	*
	Page 2 off (main memory)	\$C054	49236	-16300	*
	Read PAGE2 switch	\$C01C	49180	-16356	Read
HIRES	On: access high-res. pages	\$C057	49239	-16297	†
	Off: use RAMRD, RAMWRT	\$C056	49238	-16298	†
	Read HIRES switch	\$C01D	49181	-16355	Read
ALTZP	Auxiliary stack & z.p.	\$C009	49161	-16373	Write
	Main stack & zero page	\$C008	49160	-16374	Write
	Read ALTZP switch	\$C016	49174	-16352	Read

* When 80STORE is on, the PAGE2 switch selects main or auxiliary display memory.

† When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to switch between the high-resolution Page-1 area in main memory or auxiliary memory.

Table 4-8. 48K RAM Transfer Routines

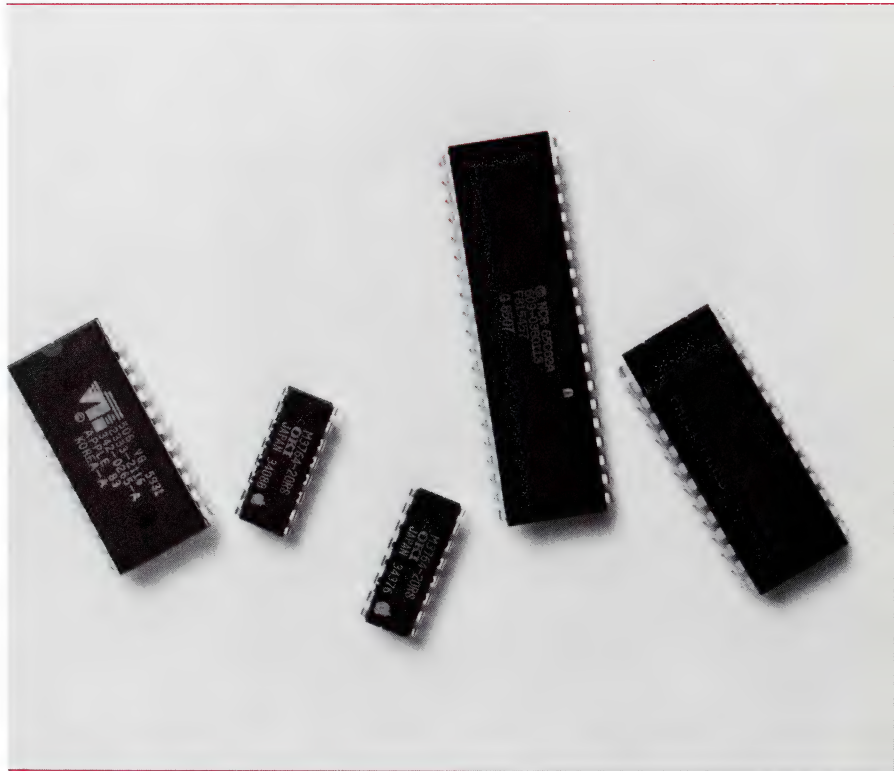
Name	Action	Hex	Function
AUXMOVE	JSR	\$C312	Moves data blocks between main and auxiliary 48K memory.
XFER	JMP	\$C314	Transfers program control between main and auxiliary 48K memory.

Table 6-5. I/O Memory Switches

Name	Function	Hex	Location		Notes
			Hex	Decimal	
SLOT3ROM	Slot ROM at \$C300	\$C00B	49163	-16373	Write
	Internal ROM at \$C300	\$C00A	49162	-16374	Write
	Read SLOT3ROM switch	\$C017	49175	-16361	Read
SLOT4ROM	Slot ROM at \$C400	\$C006	49159	-16377	Write
	Internal ROM at \$C400	\$C007	49158	-16378	Write
	Read SLOT4ROM switch	\$C015	49173	-16363	Read

Table 6-7. I/O Routine Offsets and Registers Under Pascal 1.1 Protocol

Addr.	Offset for	X Register	Y Register	A Register
\$Cs0D	Initialization			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	(unchanged)
\$Cs0E	Read			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	Character read
\$Cs0F	Write			
	On entry	\$Cs	\$s0	Char. to write
	On exit	Error code	(unchanged)	(unchanged)
\$Cs10	Status			
	On entry	\$Cs	\$s0	Request (0 or 1)
	On exit	Error code	(changed)	(unchanged)



This appendix explains how to use 80-column text cards with high-level languages. Information about using 80-column text cards with assembly language programs through the Apple IIe Monitor firmware is found in Chapter 3 of this manual. The information in this appendix applies to the Apple IIe 80-Column Text Card and the Apple IIe Extended 80-Column Text Card.

If you are using Applesoft, ProDOS, or DOS you can choose to leave the 80-column text card inactive after installing it. You will want to do this when running software that does not take advantage of the 80-column display capability.

The startup procedure for displaying 80 columns of text on your Apple IIe depends on which operating system you plan to use. Starting up the system with Apple II Pascal or CP/M® is very easy; the operating system does it for you; the procedures for starting up with ProDOS or DOS 3.3 are slightly more complicated, but not difficult.





Starting Up With Pascal or CP/M

Pascal programmers don't have to activate the text card because Pascal does it for them. If you use the Pascal language or the CP/M operating system, displaying 80 columns of text is automatic once you've installed the card. Simply start up your system with any Pascal or CP/M startup disk.

CP/M: CP/M (Control Program for Microprocessors) is a trademark of Digital Research. To use the CP/M operating system with your Apple IIe, make sure the SOFTCARD® by Microsoft or the Z-Engine™ by Advanced Logic Systems is correctly installed before you start up the computer.

Co-Processor Cards and Interrupts: Some co-processor cards that were designed for use in the Apple II Plus may not work with an Apple IIe without some modification. There could be problems if you want to use interrupts on the Apple IIe. If you are having problems with a co-processor card, check with the card's manufacturer for their recommendations.

Refer to the operating system reference manual for your version of Apple Pascal for more information.

When using Apple II Pascal 1.1, you'll probably want to run the program SETUP to make the  and  keys functional. SETUP is a self-documenting program on the Pascal disk APPLE3. Pascal versions 1.2 and later are already configured to use the  and  keys.

Starting Up With ProDOS or DOS 3.3

ProDOS and DOS 3.3 both look for a startup program on the startup (boot) disk as soon as the operating system has been loaded and begins executing. If the operating system finds the program, called STARTUP on a ProDOS disk and usually called HELLO on a DOS 3.3 disk, it will execute the program.

You can write a customized startup program that will set up the 80-column text card in any state you need. Just be sure it is on your startup disk and has the startup filename.

Here is a sample Applesoft startup program that works with both ProDOS and DOS 3.3:

```
10 HOME:D$=CHR$(4)
20 PRINT D$;"PR#3"
30 END
```

You can do whatever you wish with the program from line 20 on. Note that the screen will have switched to 80-column text mode after line 20.

By the Way: If you arrange to have the card active automatically, you will still, of course, be able to switch into 40-column mode.

Using the GET Command

The presence of an active 80-column text card in the IIe requires that BASIC programmers use some alternate to Applesoft's INPUT command if their programs are to be userproof. Applesoft programmers should use either the GET command or the RDKEY or GETLN subroutines.

This is because the escape sequences used to switch back and forth between modes or to deactivate the card sometimes make it necessary to accept escape sequences in INPUT mode when using an 80-column card. Because the program accepts escape sequences typed from the keyboard, your program will not be userproof against accidental sequences typed in response to an INPUT command.

To get around this problem, you can use the GET command instead. The program does not read escape sequences typed from the keyboard in response to a GET command. This means that your users can err in their responses without endangering the display.

When to Switch Modes Versus When to Deactivate

When using BASIC, deactivate the text card whenever a previous (BASIC) program has left the card active (leaving a solid cursor on the screen) or whenever you want to send output to a peripheral device.

Switch back and forth between 40-column and 80-column displays for visual appeal. For full use of the control characters described later, your card must be active, although it can display in either 40-column or 80-column mode.

Original Ile

Tabbing in Applesoft: You must switch to a 40-column display to use Applesoft comma tabbing or the HTAB command.

Display Features With the Text Card

With an active 80-column card you can issue BASIC and PRODOS commands in lowercase characters. You can also issue commands in lowercase from the keyboard, that is, in immediate mode. This is particularly convenient because REM statements and data within quotes remain in lowercase as they were typed.

If you are using DOS 3.3, you must issue commands in uppercase whether or not your card is active.

INVERSE, FLASH, NORMAL, HOME

There are several commands you can give your computer from Applesoft BASIC to affect the appearance of text on the screen. All of these features are described in the *Applesoft BASIC Programmer's Reference Manual*.

- INVERSE tells the computer to display black characters on a white background instead of the normal display of white characters on a black background. This command is normally only available for uppercase characters, but with an active 80-column text card it is available for uppercase and lowercase characters.
- FLASH causes subsequently printed characters to blink quickly between inverse and normal characters. You can turn off the FLASH command by typing the NORMAL command. The FLASH command is normally available only with uppercase characters; it is not available at all while the card is active.

- NORMAL tells the computer to turn off the INVERSE or FLASH command and to display subsequently printed characters normally. It works the same way with the card active or inactive.
- HOME clears the screen and returns the cursor to the upper-left corner of the screen. Both the NORMAL HOME and INVERSE HOME commands are available while the card is active, but INVERSE HOME works a little differently when the card is active.

By the Way: The FLASH and INVERSE commands can be used to highlight important screen messages within a BASIC program.

Important!

If you are using the FLASH command (which means the 80-column text card is inactive) and then type PR#3 to activate the card, the screen turns white as the cursor goes to the HOME position. Whatever you type appears in black characters on the white screen. If you list or run an Applesoft BASIC program, some of the characters will appear as MouseText characters. To avoid this, remember to use either the NORMAL or INVERSE command before you exit the program.

Tabbing With the Original Apple IIe

You cannot use conventional 40-column tabbing in BASIC with the original model Apple IIe with an 80-column display. You do not have to turn off your card, but you must switch out of 80-column mode to use the HTAB command or to use comma tabbing.

When an original Apple IIe is displaying 80-column text, you should use the POKE 1403 command for horizontal tabbing in the right half of the screen instead of the HTAB command.

Comma Tabbing With the Original Apple IIe

In BASIC you can use commas in PRINT statements to instruct the computer to display all or part of your output in columns. This is known as comma tabbing. You can use this method of tabbing as long as the screen is displaying 40 columns (that is with the card inactive or after issuing an `[ESCAPE]-[4]` command to switch to 40-column mode). You cannot use this method of tabbing with an 80-column display. If you try to do so, characters will be placed in memory outside the screen area and may change programs or data in memory.

HTAB and POKE 1403

The VTAB (vertical tab) and HTAB (horizontal tab) statements can be used to place the cursor at a specific location on the screen before printing characters. The largest value you can use with the VTAB statement is 24; the largest for HTAB is 255. The VTAB command works just the same in an 80-column display as it does in a 40-column display.

On the original Apple IIe, the HTAB command causes the cursor to wrap around to the next line after it reaches the 40th column, so you cannot use this command to position the cursor in the last 40 columns while the screen is displaying 80 columns.

POKE 1403 is specifically designed to solve this problem. Using the POKE 1403 command allows you to tab horizontally across the extra 40 columns provided by the 80-column text card.

If you want to tab past column 40 while the card is active and the screen is displaying 80 columns, use the following, where n is a number from 0 to 79:

POKE 1403, n

When you use the HTAB command, HTAB 1 places the cursor at the leftmost position on the screen. When you use the POKE 1403 command, POKE 1403,0 places the cursor at the leftmost position on the screen.

Using Control Characters With the Card

Using BASIC with an active 80-column text card increases the number of functions you can perform with control characters. Originally control-character commands were so named because they were given from the keyboard by pressing the **CONTROL** key in conjunction with another key. You can perform the same functions from your programs by using an equivalent control-character code. Commands based on these two-key combinations are called control-character commands even when they must be issued from a program.

Control Characters and Their Functions

Table G-1 lists the control-character commands supported by BASIC with an 80-column card. The table includes the corresponding command code, its function and whether a given command can be executed from the keyboard as well as from a program.

Table G-1. Control Characters With 80-Column Firmware On

Control Character	ASCII Code	Apple IIe Name	Action Taken by BASICOUT
Control-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
Control-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
Control-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
Control-K †	VT	clear EOS	Clears from cursor position to the end of the screen.
Control-L †	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.
Control-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.
Control-N †	SO	normal	Sets display format normal.
Control-O †	SI	inverse	Sets display format inverse.
Control-Q †	DC1	40-column	Sets display to 40-column.
Control-R †	DC2	80-column	Sets display to 80-column.
Control-S *	DC3	stop-list	Stops listing characters on the display until another key is pressed.
Control-U †	NAK	quit	Deactivates 80-column video firmware.
Control-V †	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.
Control-W †	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.
Control-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.

Table G-1—Continued. Control Characters With 80-Column Firmware On

Control Character	ASCII Code	Apple IIe Name	Action Taken by BASICOUT
Control-Y †	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).
Control-Z †	SUB	clear line	Clears the line the cursor position is on.
Control-[ESC	enable MouseText	Map inverse uppercase characters to MouseText characters.
Control-\ †	FS	forward space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.
Control-]†	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window).
Control-__	US	up	Moves cursor up a line, no scroll.

* Only works from the keyboard.

† Doesn't work from the keyboard.

How to Use Control-Character Codes in Programs

To issue a control-character command from a program, use the ASCII decimal code that corresponds to the control-character. (See Table G-1.)

The following example shows how to use ASCII decimal codes in an Applesoft BASIC program. Type

```
HOME [ ? ]
NEW
10 PRINT CHR$(15): PRINT "MAKE HAY"
20 PRINT CHR$(14): PRINT "WHILE THE SUN SHINES"
RUN
```

(CHR\$ is the Applesoft BASIC command that signifies that a control-character function is to be performed.)

You will get

```
JNEW
J10 PRINT CHR$(15): PRINT "MAKE HAY"
J20 PRINT CHR$(14): PRINT "WHILE THE SUN SHINES"
JRUN
MAKE HAY
WHILE THE SUN SHINES
J■
```

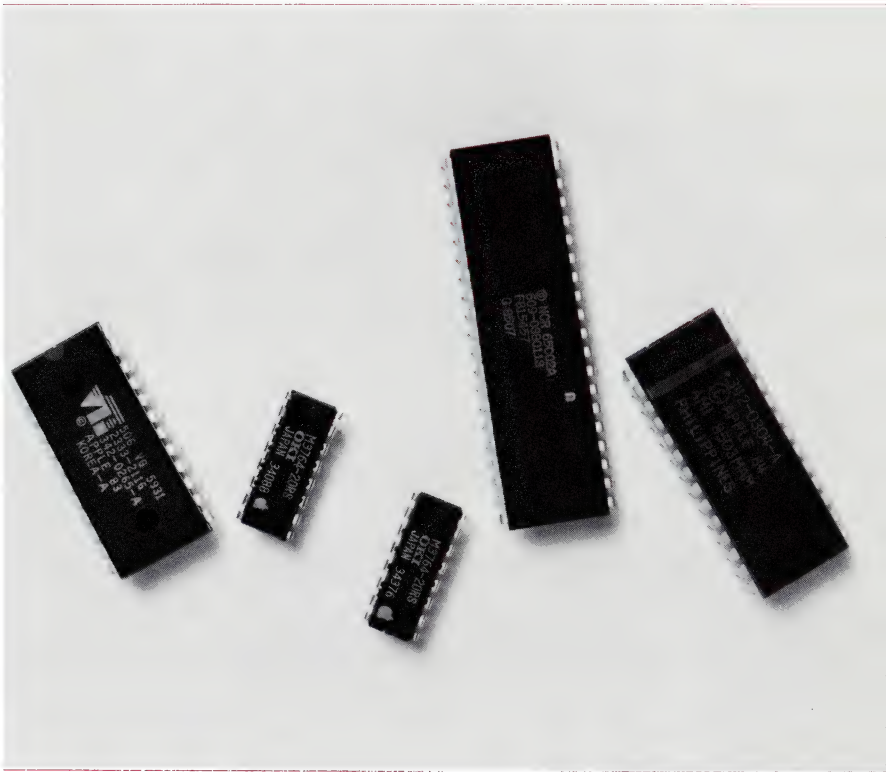
See Chapter 3 in this manual for a description of control-character functions.

The ASCII decimal codes for inverse video (Control-O) and normal video (Control-N) are 15 and 14. When the PRINT statements in the example are executed, the display switches to inverse and prints MAKE HAY, then switches back to a normal display and prints WHILE THE SUN SHINES.

A Word of Caution to Pascal Programmers

Avoid writing Control-U or Control-Q to the console from a Pascal program. Either one puts the system into a state that will cause Pascal to eventually crash.

You can't send control characters from the keyboard to the 80-column firmware when using Pascal. The only exceptions to this rule are Control-M (CR) and Control-G (BEL).



For more information about the installation and operation of the SSC, see the Super Serial Card manual.

This appendix briefly describes how to use the Apple II Super Serial Card (SSC) from programs, how to find the SSC through software, and the commands supported by the SSC.

The SCC is one of the most common serial interface cards used with the Apple IIe, and the Apple IIc's serial ports operate very much like the Super Serial Card. This similarity should make it easier for you to write programs for both the Apple IIe and Apple IIc.

Locating the Card

The Pascal 1.1 firmware protocol is described in Chapter 6.

Locations \$Cs05, \$Cs07, \$Cs0B, and \$Cs0C (where *s* is the number of the slot where the SSC is installed) contain the identification bytes for the Super Serial Card. The identification byte's values are

\$Cs05	\$38
\$Cs07	\$18
\$Cs0B	\$01
\$Cs0C	\$31

Operating Modes

The Super Serial Card has two main operating modes: printer mode and communications mode. There is nothing you can do from software to change from one mode to the other since they are set by the position of the jumper block.

Note to Software Developers: If you are writing software that depends on the SSC being in a given operating mode, make sure that your documentation tells the user to set up the SSC in the proper way.

In printer mode, the SSC is set to send data to a printer, local terminal, or other serial device. In communications mode, the SSC is set to operate with a modem. From communications mode, the SSC can enter a special mode called terminal mode. In terminal mode the Apple IIe acts like an unintelligent terminal.

Operating Commands

For each of the operating modes, you can control many aspects of data transmission such as baud rate, data format, line feed generation, and so forth.

Your program can change these aspects by sending control codes as commands to the card. All commands are preceded by a command character and followed by a carriage return character (\$0D).

The command character is usually Control-I in printer mode and Control-A in communications mode and terminal mode. In the command examples in the following sections, Control-I is used unless the command being described is available only in communications mode or terminal mode. A carriage return character is represented by its ASCII symbol, CR.

There are three types of command formats:

- A number, represented by n, followed by an uppercase letter with no space between the characters (for example, 4D to set data format 4).
- An uppercase letter by itself (for example, R to reset the SSC).
- An uppercase letter followed by a space and then either E to enable or D to disable a feature (for example, L D to disable automatic insertion of line feed characters).

The allowable range of n is given in each command description that follows.

The choice of enable or disable is indicated with E/D. The underscore character (__) before the E/D in commands that allow enable/disable is to remind you that a space is required there.

The SSC checks only numbers and the first letters of commands and options. (All such letters must be uppercase.) Further letters, which you can add to assist your memory, have no effect on the SSC. For example, XOFF Enable is the same as X E. The SSC ignores invalid commands.

Important!

The spaces in command examples are there for clarity; generally you will not use spaces in a command string. Where a space is required in a command string, an underscore (__) character will appear in the text as a reminder.

The Command Character

The normal command character is Control-I (ASCII \$09) in printer mode, or Control-A (ASCII \$01) in communications mode. If you want to change the command character from Control-I to Control-something else, send Control-I Control-something else. For example, to change the command character to Control-W, send Control-I Control-W. To change back, send Control-W Control-I. No return character is required after either of these commands.

You can send the command character itself through the SSC by sending it twice in a row: Control-I Control-I; no return character is required after this command. This special command allows you to transmit the command character without affecting the operation of the SSC, and without having to change to another command character and then back again later.

Here is how to generate this character in BASIC and Pascal:

Applesoft BASIC: **PRINT CHR\$(9); "command"**
Pascal: **WRITELN (CHR(9), 'command');**

Baud Rate, nB

You can use this command to override the physical settings of switches SW1-1 through SW1-4 on the SSC. For example, to change the baud rate to 135, send Control-I 4B CR to the SSC.

Table H-1. Baud Rate Selections

n	SSC Baud Rate	n	SSC Baud Rate
0	use SW1-1 to SW1-4	8	1200
1	50	9	1800
2	75	10	2400
3	109.92 (110)	11	3600
4	134.58 (135)	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200

Data Format, nD

You can override the settings of switch SW2-1 with this command. The table below shows how many data and stop bits correspond to each value of n. For example, Control-I 2D CR makes the SSC transmit each character in the form one start bit (always transmitted), six data bits, and one stop bit.

Table H-2. Data Format Selections

n	Data Bits	Stop Bits
0	8	1
1	7	1
2	6	1
3	5	1
4	8	2 *
5	7	2
6	6	2
7	5	2 †

* 1 with Parity options 4 through 7

† 1½ with Parity options 0 through 3

Parity, nP

You can use this command to set the parity that you want to use for data transmission and reception. There are five parity options available, described in Table H-3.

Table H-3. Parity Selections

n	Parity to Use
0, 2, 4 or 6	None (default value)
1	Odd parity (odd total number of ones)
3	Even parity (even total number of ones)
5	MARK parity (parity bit always 1)
7	SPACE parity (parity bit always 0)

For example, the command string Control-I IP CR makes the SSC transmit and check for odd parity. Odd parity means that the high bit of every character is 0 if there is an odd number of 1 bits in that character, or 1 if there is an even number of 1 bits in the character, making the total number of 1 bits in the character always odd. This is an easy (but not foolproof) way to check data for transmission errors. Parity errors are recorded in a status byte.

Set Time Delay, nC, nL, and nF

Some printers can't keep up with the Apple IIe when they are doing certain operations. You may need to change default settings on the SSC to give a printer the time it needs.

The nC command overrides the setting of switch SW2-2 on the SSC. That switch provides two choices: either no delay or a 250 millisecond delay after the SSC sends a carriage return character.

The nL command allows time after a line feed character for a printer platen to turn so the paper is vertically positioned to receive the next line.

The nF command allows time after a form feed character for the printer platen to move the paper form to the top of the next page (typically a longer time than a line feed).

Table H-4. Time Delay Selections

n	Time Delay
0	none
1	32 milliseconds
2	250 milliseconds (1/4 second)
3	2 seconds

Consult the user manual for a given printer to find out how much time it takes to move its print head and platen so you can determine an appropriate set of values for these three delays. The idea is to have at least enough time for the printer parts to move the required distance, but not so much time that overall printing speed is slowed down drastically. Many printers require no delays because they have a buffer built in to keep accepting characters even while they are doing form feeds and so on.

A typical setup for a *very* slow printer would be Control-I 2C CR, Control-I 2L CR, Control-I 3F CR; that is, the SSC waits 250 milliseconds after transmitting carriage returns, 250 milliseconds after transmitting line feeds, and 2 seconds after transmitting form feed characters.

Echo Characters to the Screen, E_E/D

For the Apple IIe, as for most computers, displaying (echoing) a character on the video screen during communications is a separate step from receiving it from the keyboard, though we tend to think of these as one step, as on a typewriter. For example, if you send Control-A E_D CR, the SSC does not forward incoming characters to the Apple IIe screen. This can be used to hide someone's password entered at a terminal, or to avoid double display of characters.

This command is used in communications mode only.

Automatic Carriage Return, C

Sending Control-I C CR to the SSC causes it to generate a carriage return character (ASCII CR) whenever the column count exceeds the current printer line width limit. This command is used in printer mode only.

Important!

Once this option is on, only clearing the high-order bit at location \$578+s (where *s* is the slot the SSC is in) can turn this option back off. This option is normally off.

Automatic Line Feed, L_E/D

You can use this command to have the SSC automatically generate and transmit a line feed character after each carriage return character. This overrides the setting of switch SW2-5. For example, send Control-I L_E CR to your printer to print listings or double-spaced manuscripts for editing.

Mask Line Feed In, M_E/D

If you send Control-I M_E CR to the SSC, it will ignore any incoming line feed character that immediately follows a carriage return character.

Reset Card, R

Sending Control-I R CR to the SSC has the same effect as sending a PR#0 and an IN#0 to a BASIC program and then resetting the SSC. This command cancels all previous commands to the SSC and puts the physical switch settings back into force.

Specify Screen Slot, S

In communications mode, you can specify the slot number of the device where you want text or listings displayed with this command. (Normally this is slot 0, the Apple IIe video screen.) This allows chaining of the SSC to another card slot, such as an 80-column text card. For the firmware in the SSC to pass on information to the firmware in the other card, the other card must have an output entry point within its \$Cs00 space; this is the case for all currently available 80-column cards for the Apple IIe.

For example, let's say you have the SSC in slot 2 with a remote terminal connected to it, and an 80-column card in slot 3. Send Control-A S3 CR to cause the data from the remote terminal to be chained through the card in slot 3, so that it is displayed on the Apple IIe in 80-column format. (Not available in Pascal.)

Translate Lowercase Characters, nT

The Apple IIe Monitor translates all incoming lowercase characters into uppercase ones before sending them to the video screen or to a BASIC program. The nT command has four options, which are shown in Table H-5.

Table H-5. Lowercase Character Display Options

n	Action
0	Change all lowercase characters to uppercase ones before passing them to a BASIC program or to the video screen. This is the way the Apple IIe monitor handles lowercase.
1	Pass along all lowercase characters unchanged. The appearance of the lowercase characters on the Apple II screen is undefined (garbage).
2	Display lowercase characters as uppercase inverse characters (that is, as black characters on a white background).
3	Pass lowercase characters to programs unchanged, but display lowercase as uppercase, and uppercase as inverse uppercase (that is, as black characters on a white background).

Suppress Control Characters, Z

If you issue the Z command described here, all further commands are ignored; this is useful if the data you are transmitting, such as graphics data, contains bit patterns that the SSC can mistake for control characters.

Sending Control-I Z CR to the SSC prevents it from recognizing any further control characters (and hence commands) whether coming from the keyboard or contained in a stream of characters sent to the SSC.

Important!

The only way to reinstate command recognition after the Z command is to either reinitialize the SSC, or clear the high-order bit at location \$5F8+s (where *s* is the number of the slot in which the SSC is installed).

Find Keyboard, F_E/D

You can use this command to make the SSC ignore keyboard input.

For example, you can include Control-I F_D CR in a program, followed by a routine that retrieves data through the SSC, followed by Control-I F_E CR to turn the keyboard back on.

XOFF Recognition, X_E/D

Sending Control-I X_E CR to the SSC causes it to look for any XOFF (\$13) character coming from a device attached to the SSC, and to respond to it by halting transmission of characters until the SSC receives an XON (\$11) from the device, signalling the SCC to continue transmission. In printer mode, this function is normally turned off.

Caution

In printer mode, full duplex communication may not work with XOFF recognition turned on, so be careful.

Tab in BASIC, T_E/D

In printer mode only, if you send Control-I T_E CR to the SSC, the BASIC horizontal position counter is left equal to the column count. All tabs work, including back-tabs. Tabs beyond column 40 require a POKE to location 36. Commas only work as far as column 40, and BASIC programs will be listed in 40-column format.

Note that this use of tabbing is specific to the SSC—it doesn't go through the 80-column firmware.

Terminal Mode

From communications mode, the SSC can enter terminal mode and make the Apple IIe act like an unintelligent terminal. This is useful for connecting the Apple IIe to a computer timesharing service, or for conversing with another Apple II.

Entering Terminal Mode, T

Send Control-A T CR to enter terminal mode. This causes the Apple IIe to function as a full-duplex unintelligent terminal. You can use this command together with the Echo command to simulate the half-duplex terminal mode of the old Apple II Communications Card.

By the Way: If you enter terminal mode and don't see what you type echoed on the Apple video screen, probably the modem link has not yet been established, or you need to use the Echo Enable command (Control-A E_E CR).

Transmitting a Break, B

Sending Control-A B CR causes the SSC to transmit a 233-millisecond break signal, recognized by most time-sharing systems as a signoff.

Special Characters, S_E/D

If you send Control-A S__D CR, the SSC will treat the `ESCAPE` key like any other key.

Quitting Terminal Mode, Q

Send Control-A Q CR to the SSC to exit from terminal mode.

SSC Error Codes

The SSC uses I/O scratchpad address \$678+s (*s* is the number of the slot that the SSC is in) to record status after a read operation. The firmware calls this byte STSBYTE. Table H-6 lists the bit definitions of this byte.

Table H-6. STSBYTE Bit Definitions

Bit	“1” Means	“0” Means
0	Parity Error occurred.	No Parity Error occurred.
1	Framing Error occurred.	No Framing Error occurred.
2	Overrun occurred.	No Overrun occurred.
3	Carrier lost.	Carrier present.
5	Error occurred.	No error occurred.

The terms **Parity**, **Framing Error**, and **Overrun** are defined in the glossary.

Bits 0, 1, and 2 are the same as the corresponding three bits of the ACIA Status Register of the SSC. Bit 3 indicates whether or not the Data Carrier Detect (DCD) signal went false at any time during the receive operation.

Bit 5 is set if any of the other bits are set, as an overall error indicator. If bit 5 is the only bit set, an unrecognized command was detected. If all bits are 0, no error occurred.

These error codes begin with the number 32 to avoid conflicting with previously defined and documented system error codes.

In BASIC, you can check this status byte via a PEEK \$678+s (*s* is the SSC slot), and reset it with a POKE command at the same location.

In Pascal, the IORESULT function returns the error code value.

By the Way: Any character—including the carriage return at the end of a WRITELN statement—will cause posting of a new value in IORESULT.

Table H-7 shows the possible combinations of error bits corresponding to these decimal error codes.

Table H-7. Error Codes and Bits

Error Code*	Carrier Lost	Overrun	Framing Error	Parity Error
0		no error		
32		illegal command		
33	no	no	no	yes
34	no	no	yes	no
35	no	no	yes	yes
36	no	yes	no	no
37	no	yes	no	yes
38	no	yes	yes	no
39	no	yes	yes	yes
40	yes	no	no	no
41	yes	no	no	yes
42	yes	no	yes	no
43	yes	no	yes	yes
44	yes	yes	no	no
45	yes	yes	no	yes
46	yes	yes	yes	no
47	yes	yes	yes	yes

* Result of PEEK \$678+s in BASIC or IORESULT in Pascal.

The ACIA

The Asynchronous Communication Interface Adapter (ACIA) chip is the heart of the Super Serial Card. It takes the 1.8432 MHz signal generated by the crystal oscillator on the SSC and divides it down to one of the fifteen baud rates that it supports. The ACIA also handles all incoming and outgoing signals of the RS232-C serial protocol that the ACIA supports.

The ACIA registers control hardware handshaking and select the baud rate, data format, and parity. The ACIA also performs parallel to serial and serial to parallel data conversion, and buffers data transfers.

SSC Firmware Memory Use

Table H-8 is an overall map of the locations that the SSC uses, both in the Apple IIe and in the SSC's own firmware address space.

Table H-8. Memory Use Map

Address	Name of Area	Contents
\$0000-\$00FF	Page zero	Monitor pointers, I/O hooks, and temporary storage.
\$04xx-\$07xx (selected locations)	Peripheral slot Scratchpad RAM	Locations (8 per slot) in Apple IIe pages \$04 through \$07. SSC uses all 8 of them.
\$C0(8+s)0- \$C0(8+s)F	Peripheral card I/O space	Locations (16 per slot) for general I/O; SSC uses 6 bytes.
\$Cs00-\$CsFF	Peripheral card ROM space	One 256-byte page reserved for card in slot s; first page of SSC firmware.
\$C800-\$CFFF	Expansion ROM	Eight 256-byte pages reserved for 2K ROM or PROM; SSC maps its firmware onto \$C800-\$CEFF.

Zero-Page Locations

The SSC uses the zero-page locations described in Table H-9.

Table H-9. Zero-Page Locations Used by the SSC

Address	Name	Description
\$24 *	CH	Monitor pointer to current position of cursor on screen
\$26	SLOT16	Usually (slot x 16); that is, \$s0
\$27	CHARACTER	Input or output character
\$28 *	BASL	Monitor pointer to current screen line
\$2A	ZPTMP1	Temporary storage (various uses)
\$2B	ZPTMP2	Temporary storage (various uses)
\$35	ZPTMP	Temporary storage (various uses)
\$36 *	CSWL	BASIC output hook (not for Pascal)
\$37 *	CSWH	High byte of CSW
\$38 *	KSWL	BASIC input hook (not for Pascal)
\$39 *	KSWH	High byte of KSW
\$4E *	RNDL	Random number location, updated when looking for a keypress (not used when initialized by Pascal)

* Not used when Pascal initializes SSC.

Peripheral Card I/O Space

There are 16 bytes of I/O space allocated to each slot in the Apple IIe. Each set begins at address \$C080 + (slot x 16); for example, if the SSC is in slot 3, its group of bytes extends from \$C0B0 to \$C0BF. Table H-10 interprets the 6 bytes the SSC uses.

Table H-10. Address Register Bits Interpretation

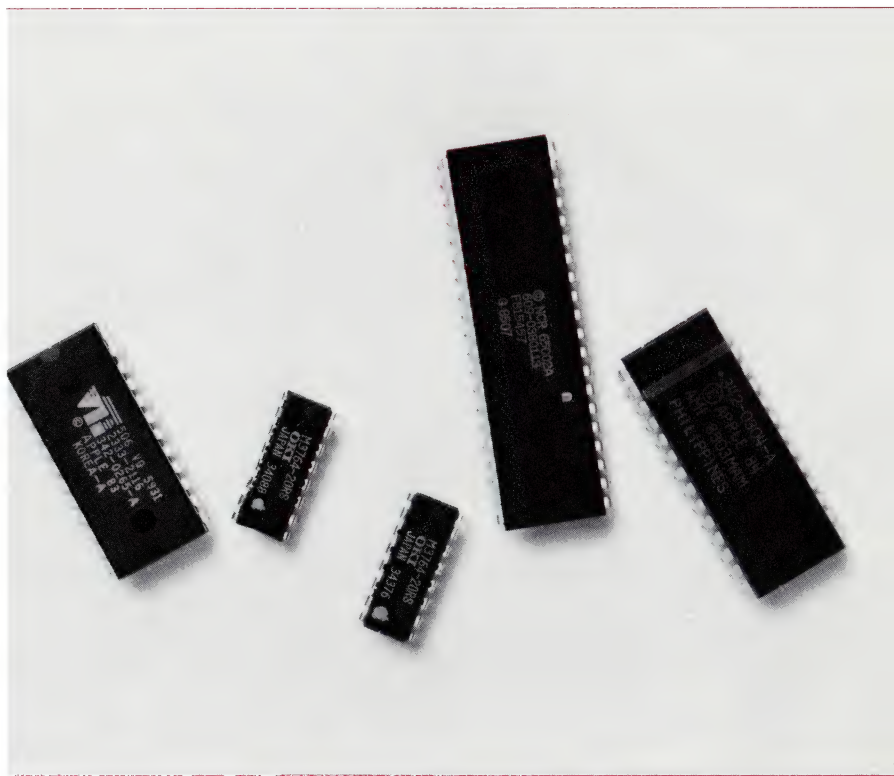
Address	Register	Bits	Interpretation
\$C081+s0	DIPSW1 (SW1-x)	0	SW1-6 is OFF when 1, ON when 0
		1	SW1-5 is OFF when 1, ON when 0
		4-7	Same as above for SW1-4 through SW1-1
\$C082+s0	DIPSW2 (SW2-x)	0	Clear To Send (CTS) is true when 0
		1-3	Same as above for SW2-5 through SW2-3
		5, 7	Same as above for SW2-2 and SW2-1
\$C088+s0	TDREG RDREG	0-7	ACIA transmit register (write)
		0-7	ACIA receive register (read)
\$C089+s0	STATUS		ACIA status/reset register
		0	Parity error detected when 1
		1	Framing error detected when 1
		2	Overrun detected when 1
		3	ACIA receive register full when 1
		4	ACIA transmit register empty when 1
		5	Data Carrier Detect (DCD) true when 0
		6	Data Set Ready (DSR) true when 0
		7	Interrupt (IRQ) has occurred when 1
\$C08A+s0	COMMAND		ACIA command register (read/write)
		0	Data Terminal Ready (DTR): enable (1) or disable (0) receiver and all interrupts
		1	When 1, allow STATUS bit 3 to cause interrupt
		2-3	Control transmit interrupt, Request To Send (RTS) level, and transmitter
		4	When 0, normal mode for receiver; when 1, echo mode (but bits 2 and 3 must be 0)
		5-7	Control parity
\$C08B+s0	CONTROL		ACIA control register (read/write)
		0-3	Baud rate: \$00 = 16 times external clock; See Table H-1.
		4	When 1, use baud rate generator; when 0, use external clock (not supported)
		5-6	Number of data bits: 8 (bit 5 and 6 = 0) 7 (5 = 1, 6 = 0), 6 (5 = 0, 6 = 1) or 5 (bit 5 and 6 both = 1)
		7	Number of stop bits: 1 if bit 7 = 0; if bit 7 = 1, then 1-1/2 (with 5 data bits, no parity), 1 (8 data plus parity), or 2

Scratchpad RAM Locations

The SSC uses the scratchpad RAM locations listed in Table H-11.

Table H-11. Scratchpad RAM Locations Used by the SSC

Address	Field name	Bit	Interpretation
\$0478+s	DELAYFLG	0-1	Form feed delay selection
		2-3	Line feed delay selection
		4-5	Carriage return delay selection
		6-7	Translate option
\$04F8+s	PARAMETE	0-7	Accumulator for firmware's command processor
\$0578+s	STATEFLG	0-2	Command mode when not 0
		3-5	Slot to chain to (communications mode)
		6	Set to 1 after lowercase input character
		7	Terminal mode when 1 (communications mode)
\$05F8+s	CMDBYTE	7	Enable CR generation when 1 (printer mode)
		0-6	Printer mode default is Control-I; communications mode default is Control-A
		7	Set to 1 to Zap control commands
\$0678+s	STSBYTE		Status and IORESULT byte
\$06F8+s	CHNBYTE	0-2	Current screen slot (communication mode); when slot = 0, chaining is enabled.
		3-7	\$Cs00 space entry point (communications mode)
	PWDBYTE	0-7	Current printer width; for listing compensation, auto-CR (printer mode)
\$0778+s	BUFBYTE	0-6	One-byte input buffer (communications mode); used in conjunction with XOFF recognition
		7	Set to 1 when buffer full (communications mode)
	COLBYTE	0-7	Current-column counter for tabbing and so forth (printer mode)
\$07F8+s	MISCFLG	0	Generate line feed after CR when 1
		1	Printer mode when 0; communications mode when 1
		2	Keyboard input enabled when 1
		3	Control-S (XOFF), Control-R, and Control-T input checking when 1
		4	Pascal operating system when 1; BASIC when 0
		5	Discard line feed input when 1
		6	Enable lowercase and special character generation when 1 (communications mode)
		6	Tabbing option on when 1 (printer mode)
		7	Echo output to Apple IIe screen when 1



```

00:      0000      1 TEST      EQU 0      :REAL VERSION

0000:          2          LST ON          ;DO LISTING AND SYMBOL TABLES
0000:          3          MSB ON          ;SET THEM HIBITS
0000:      0001      4 IROTEST EQU 1
0000:      0000      5          DO TEST
S          6 F8ORG EQU $1800
S          7 C1ORG EQU $2100
S          8 C3ORG EQU $2300
S          9 C8ORG EQU $2800
0000:      10          ELSE
0000:      F800      11 F8ORG EQU $F800
0000:      C100      12 C1ORG EQU $C100
0000:      C300      13 C3ORG EQU $C300
0000:      C800      14 C8ORG EQU $C800
0000:      15          FIN
0000:      16          MSB ON
0000:      17          INCLUDE EQUATES
0000:      1 *****
0000:      2 *
0000:      3 * Apple //e Video Firmware
0000:      4 *
0000:      5 * RICK AURICCHIO 08/81
0000:      6 * E. BEERNINK, R. WILLIAMS 1984
0000:      7 *
0000:      8 * (C) 1981,1984 APPLE COMPUTER INC.
0000:      9 * ALL RIGHTS RESERVED
0000:     10 *
0000:     11 *****
0000:     12 *
0000:     0006     13 GOODF8 EQU 6          ;F8 ROM VERSION
0000:     14 *
0000:     15 * HARDWARE EQUATES:
0000:     16 *
0000:     C000     17 KBD EQU $C000          ;Read keyboard
0000:     C000     18 CLR80COL EQU $C000      ;Disable 80 column store
0000:     C001     19 SET80COL EQU $C001      ;Enable 80 column store
0000:     C002     20 RDMAINRAM EQU $C002      ;Read from main RAM
0000:     C003     21 RDCARDRAM EQU $C003      ;Read from auxiliary RAM
0000:     C004     22 WRMAINRAM EQU $C004      ;Write to main RAM
0000:     C005     23 WRCARDRAM EQU $C005      ;Write to auxiliary RAM
0000:     C006     24 SETSL0TCXROM EQU $C006   ;Switch in slot CX00 ROM
0000:     C007     25 SETINTCXROM EQU $C007   ;Switch in internal CX00 ROM
0000:     C008     26 SETSTDZP EQU $C008      ;Switch in main stack/zp/lang.card
0000:     C009     27 SETALTZP EQU $C009      ;Switch in aux stack/zp/lang.card
0000:     C00A     28 SETINTC3ROM EQU $C00A    ;Switch in internal $C3 ROM
0000:     C00B     29 SETSL0TC3ROM EQU $C00B   ;Switch in slot $C3 space
0000:     C00C     30 CLR80VID EQU $C00C      ;Disable 80 column video
0000:     C00D     31 SET80VID EQU $C00D      ;Enable 80 column video
0000:     C00E     32 CLRALTCHAR EQU $C00E     ;Normal Apple II char set
0000:     C00F     33 SETALTCHAR EQU $C00F     ;Norm/inv LC, no flash
0000:     C010     34 KBDSTRB EQU $C010       ;Clear keyboard strobe
0000:     C011     35 RDLCBNK2 EQU $C011      ;>127 if LC BANK2 in use
0000:     C012     36 RDLGRAM EQU $C012      ;>127 if LC is read enabled

```

```

0000: C013 37 RDRAMRD EQU $C013 ;>127 if main RAM read enabled
0000: C014 38 RDRAMWRT EQU $C014 ;>127 if main RAM write enabled
0000: C015 39 RDCXROM EQU $C015 ;>127 if ROM CX space enabled
0000: C016 40 RDAITZP EQU $C016 ;>127 if alt. zp & lc enabled
0000: C017 41 RDC3ROM EQU $C017 ;>127 if slot C3 space enabled
0000: C018 42 RD80COL EQU $C018 ;>127 if 80 column store enabled
0000: C019 43 RDVBLBAR EQU $C019 ;>127 if not vertical blanking
0000: C01A 44 RDTEXT EQU $C01A ;>127 if text mode
0000: C01C 45 RDPAGE2 EQU $C01C ;>127 if page 2
0000: C01E 46 ALTCHARSET EQU $C01E ;>127 if alt char set switched in
0000: C01F 47 RD80VID EQU $C01F ;>127 if 80 column video enabled
0000: C030 48 SPKR EQU $C030 ;toggle speaker
0000: C054 49 TXTPAGE1 EQU $C054 ;switches in text page 1
0000: C055 50 TXTPAGE2 EQU $C055 ;switches in text page 2
0000: C05D 51 CLRAN2 EQU $C05D ;annunciator 2
0000: C05F 52 CLRAN3 EQU $C05F ;annunciator 3
0000: C061 53 BUTN0 EQU $C061 ;open-apple key
0000: C062 54 BUTN1 EQU $C062 ;closed-apple key
0000: C081 55 ROMIN EQU $C081 ;swap in D000-FFFF ROM
0000: C083 56 LCBANK2 EQU $C083 ;swap in LC bank 2
0000: C08B 57 LCBANK1 EQU $C08B ;swap in LC bank 1
0000: 58 *
0000: 59 * MONITOR EQUATES:
0000: 60 *
0000: FBB3 61 F8VERSION EQU F8ORG+$3B3 ;F8 ROM ID
0000: FDB3 62 KEYIN EQU F8ORG+$51B ;normal input
0000: FDF0 63 COUT1 EQU F8ORG+$5F0 ;normal output
0000: FF69 64 MONZ EQU F8ORG+$769 ;monitor entry point
0000: 65 *
0000: 66 * ZEROPAGE EQUATES:
0000: 67 *
0000: 0000 68 LOCO EQU 0 ;used for doing PR#
0000: 0001 69 LOC1 EQU 1 ;used for doing PR#
0000: 70 DSECT
0020: 0020 71 ORG $20
0020: 0001 72 WNDLFT DS 1 ;scrolling window left
0021: 0001 73 WNDWDTH DS 1 ;scrolling window width
0022: 0001 74 WNDTOP DS 1 ;scrolling window top
0023: 0001 75 WNDBTM DS 1 ;scrolling window bottom+1
0024: 0001 76 CH DS 1 ;cursor horizontal
0025: 0001 77 CV DS 1 ;cursor vertical
0026: 0002 78 DS 2 ;GBASL,GBASH
0028: 0002 79 BASL DS 2 ;points to current line of text
002A: 0029 80 BASH EQU BASL+1
002A: 0002 81 BAS2L DS 2 ;pointer used for scroll
002C: 002B 82 BAS2H EQU BAS2L+1
002C: 83 *
002F: 002F 84 ORG $2F
002F: 0001 85 LENGTH DS 1 ;length for mnemonics
0030: 0002 86 DS 2
0032: 0001 87 INVFLG DS 1 ;>127=normal, <127=inverse
0033: 0001 88 PROMPT DS 1 ;used by monitor upshift
0034: 0001 89 YSAV DS 1 ;input buffer index for mini
0035: 0001 90 SAVY1 DS 1 ;for restoring Y

```

```

0036:      0002  91 CSWL   DS    2           ;hook for output routine
0038:      0037  92 CSWH   EQU   CSWL+1
0038:      0002  93 KSWL   DS    2           ;hook for input routine
003A:      0039  94 KSWH   EQU   KSWL+1
003C:      003C  95       ORG   $3C
003C:      0002  96 A1L    DS    2           ;Monitor temps for MOVE
003E:      003D  97 A1H    EQU   A1L+1
003E:      0002  98 A2L    DS    2
0040:      003F  99 A2H    EQU   A2L+1
0040:      0002  100       DS    2           ;A3 NOT USED
0042:      0002  101 A4L    DS    2
0044:      0043  102 A4H    EQU   A4L+1
0044:      0001  103 MACSTAT DS    1           ;machine state on breaks
004E:      004E  104       ORG   $4E
004E:      0002  105 RNDL   DS    2           ;random number seed
0050:      004F  106 RNDH   EQU   RNDL+1
0000:      107       DEND
0000:      108 *
0000:      0200  109 BUF     EQU   $200         ;input buffer
0000:      110 * Permanent data in screenholes
0000:      111 *
0000:      112 * Note: these screenholes are only used by
0000:      113 * the 80 column firmware if an 80 column card
0000:      114 * is detected or if the user explicitly activates
0000:      115 * the firmware. If the 80 column card is not
0000:      116 * present, only MODE is trashed on RESET.
0000:      117 *
0000:      118 * The success of these routines rely on the
0000:      119 * fact that if 80 column store is on (as it
0000:      120 * normally is during 80 column operation), that
0000:      121 * text page 1 is switched in. Do not call the
0000:      122 * video firmware if video page 2 is switched in!!
0000:      123 *
0000:      07F8  124 MSLOT   EQU   $7F8           ;=$Cn ;n=slot using $C800
0000:      125 *
0000:      047B  126 OLDCH   EQU   $478+3       ;LAST CH used by video firmware
0000:      04FB  127 MODE    EQU   $4F8+3       ;video firmware operating mode
0000:      057B  128 OURCH   EQU   $578+3       ;80 column CH
0000:      05FB  129 OURCV   EQU   $5F8+3       ;80 column CV
0000:      067B  130 CHAR    EQU   $678+3       ;character to be printed/read
0000:      06FB  131 XCOORD   EQU   $6F8+3       ;GOTOXY X-coord (pascal only)
0000:      077B  132 TEMP1    EQU   $778+3       ;temp
0000:      077B  133 OLDBASL  EQU   $778+3       ;last BASL (pascal only)
0000:      07FB  134 TEMP2    EQU   $7F8+3       ;temp
0000:      07FB  135 OLDBASH  EQU   $7F8+3       ;last BASH (pascal only)
0000:      136 *
0000:      137 * BASIC MODE BITS
0000:      138 *
0000:      139 * 0..... - BASIC active
0000:      140 * 1..... - Pascal active
0000:      141 * .0..... -
0000:      142 * .1..... -
0000:      143 * ..0..... - Print control characters
0000:      144 * ..1..... - Don't print ctrl chars.

```

```

0000:      145 * ...0.... -
0000:      146 * ...1.... -
0000:      147 * ....0.... - Print control characters
0000:      148 * ....1.... - Don't print next ctrl char
0000:      149 * .....0.. -
0000:      150 * .....1.. -
0000:      151 * .....0.. -
0000:      152 * .....1.. -
0000:      153 * .....0 - Mouse text inactive
0000:      154 * .....1 - Mouse text active
0000:      155 *
0000: 0040 156 M.6      EQU  $40
0000: 0020 157 M.CTL2   EQU  $20      ;Don't print controls
0000: 0010 158 M.4      EQU  $10
0000: 0008 159 M.CTL    EQU  $08      ;Temp ctrl disable
0000: 0004 160 M.2      EQU  $04
0000: 0002 161 M.1      EQU  $02
0000: 0001 162 M.MOUSE   EQU  $01
0000:      163 *
0000:      164 * Pascal Mode Bits
0000:      165 *
0000:      166 * 0..... - BASIC active
0000:      167 * 1..... - Pascal active
0000:      168 * .0..... -
0000:      169 * .1..... -
0000:      170 * ..0..... -
0000:      171 * ..1..... -
0000:      172 * ...0.... - Cursor always on
0000:      173 * ...1.... - Cursor always off
0000:      174 * ....0.... - GOTOXY n/a
0000:      175 * ....1.... - GOTOXY in progress
0000:      176 * .....0.. - Normal Video
0000:      177 * .....1.. - Inverse Video
0000:      178 * .....0.. - PASCAL 1.1 F/W ACTIVE
0000:      179 * .....1.. - PASCAL 1.0 INTERFACE
0000:      180 * .....0.. - Mouse text inactive
0000:      181 * .....1.. - Mouse text active
0000:      182 *
0000: 0080 183 M.PASCAL EQU  $80      ;Pascal active
0000: 0010 184 M.CURSOR EQU  $10      ;Don't print cursor
0000: 0008 185 M.GOXY   EQU  $08      ;GOTOXY IN PROGRESS
0000: 0004 186 M.VMODE  EQU  $04      ;PASCAL VIDEO MODE
0000: 0002 187 M.PAS1.0 EQU  $02      ;PASCAL 1.0 MODE
0000:      188 *
0000:      189 * F8 ROM entries
0000:      190 *
0000: FA47 191 NEWBREAK EQU  F8ORG+$247
0000: FC74 192 IRQUSER  EQU  F8ORG+$474
0000: FC7A 193 IRQDONE2 EQU  F8ORG+$47A
0000: F8B7 194 TSTROM   EQU  F8ORG+$B7
0000:      18      INCLUDE BFUNC
----- NEXT OBJECT FILE NAME IS REFLIST.0
C100:      C100      1      ORG  C1ORG
C100:      C100      2  BFUNCPG EQU  *

```

```

C100:      FEC5      3 FUNCEXIT EQU F8ORG+$6C5 ;RETURN ADDRESS
C100:      FCFO      4 MINI      EQU  F8ORG+$4F0
C100:      5 *
C100:      6 * BASIC FUNCTION HOOK:
C100:      7 *
C100:      8 * $C100 is called by the patched $F8 ROM.
C100:      9 * It provides an extension to $F8 routines
C100:     10 * that do not work in 80 columns.
C100:     11 *
C100:     12 * Before jumping here, the $F8 rom disabled
C100:     13 * slot I/O and enabled ROM I/O. This makes
C100:     14 * the entire space from $C100 - $CFFF with the
C100:     15 * exception of the $C300 page available.
C100:     16 *
C100:     17 * On exit slot I/O is restored if necessary.
C100:     18 *
C100:     19 * INPUT: Y=FUNCTION AS FOLLOWS:
C100:     20 *
C100:     21 *           1 = KEYIN
C100:     22 *           2 = Fix escape char
C100:     23 *           3 = BASCALC
C100:     24 *           4 = VTAB or VTABZ
C100:     25 *           5 = HOME
C100:     26 *           6 = SCROLL
C100:     27 *           7 = CLREOL
C100:     28 *           8 = CLREOLZ
C100:     29 *           9 = RESET
C100:     30 *           A = CLREOP
C100:     31 *           B = RDKEY
C100:     32 *           C = SETWND
C100:     33 *           D = Mini Assembler
C100:     34 *           E = set 40 columns on PR#0/IN#0
C100:     35 *           F = Fix pick for monitor
C100:     36 *
C100:     37 * Stack has PHP for status of internal $CNO0 ROM
C100:     38 *
C100:     39 * Note: If 80 Vid is on and the MODE byte is valid,
C100:     40 * this call will be dispatched to an 80 column routine
C100:     41 * by B.FUNCO. Otherwise it will be dispatched to a
C100:     42 * 40 column routine by B.OLDFUNC. In all cases return
C100:     43 * to the Autostart ROM is done through F.RETURN.
C100:     44 *
C100:4C 13 C2      45 B.FUNC  JMP  DISPATCH ;figure out what to do
C103:      46 *
C103:A4 24      47 F.CLREOP LDY CH          ; ESC F IS CLR TO END OF PAGE
C105:A5 25      48         LDA  CV
C107:48         49 CLEOP1  PHA
C108:20 03 CE    50         JSR  VTABZ
C10B:20 F4 C1    51         JSR  X.CLREOLZ
C10E:A0 00      52         LDY  #$00
C110:68         53         PLA
C111:69 00      54         ADC  #$00      ;(carry set)
C113:C5 23      55         CMP  WNDBTM
C115:90 F0      56         BCC  CLEOP1
C107

```

```

C117:B0 34    C14D 57          BCS  GVTZ          ;=>always to VTABZ
C119:         58 *
C119:A5 22    59 F.HOME  LDA  WNDTOP
C11B:85 25    60          STA  CV
C11D:A0 00    61          LDY  #$00
C11F:84 24    62          STY  CH
C121:F0 E4    C107 63          BEQ  CLEOP1          ;(ALWAYS TAKEN)
C123:         64 *
C123:A5 22    65 F.SCROLL LDA  WNDTOP
C125:48       66          PHA
C126:20 03 CE 67          JSR  VTABZ
C129:A5 28    68 SCRL1  LDA  BASL
C12B:85 2A    69          STA  BAS2L
C12D:A5 29    70          LDA  BASH
C12F:85 2B    71          STA  BAS2H
C131:A4 21    72          LDY  WNDWDTH
C133:88       73          DEY
C134:68       74          PLA
C135:69 01    75          ADC  #$01
C137:C5 23    76          CMP  WNDBTM
C139:B0 0D    C148 77          BCS  SCRL3
C13B:48       78          PHA
C13C:20 03 CE 79          JSR  VTABZ
C13F:B1 28    80 SCRL2  LDA  (BASL),Y
C141:91 2A    81          STA  (BAS2L),Y
C143:88       82          DEY
C144:10 F9    C13F 83          BPL  SCRL2
C146:30 E1    C129 84          BMI  SCRL1
C148:A0 00    85 SCRL3  LDY  #$00
C14A:20 F4 C1 86          JSR  X.CLREOLZ
C14D:A5 25    87 GVTZ   LDA  CV
C14F:4C 03 CE 88 GVTZ2  JMP  VTABZ          ;set vertical base
C152:         89 *
C152:         C152 90 F.SETWND EQU *
C152:A9 28    91          LDA  #40
C154:85 21    92          STA  WNDWDTH
C156:A9 18    93          LDA  #24
C158:85 23    94          STA  WNDBTM
C15A:A9 17    95          LDA  #23
C15C:85 25    96          STA  CV
C15E:D0 EF    C14F 97          BNE  GVTZ2          ;=>go do vtab, exit
C160:         98 *
C160:         99 * Load Y from BAS2L and clear line
C160:        100 *
C160:A4 2A    101 F.CLREOLZ LDY BAS2L          ;set up by $F8 ROM
C162:4C F4 C1 102          JMP  X.CLREOLZ ;and clear line
C165:        103 *
C165:        104 * 80 column routines begin here
C165:        105 *
C165:4C EB CB 106 B.SCROLL JMP SCROLLUP ;DO IT FOR CALLER
C168:        107 *
C168:        108 * Clear to end of line using Y = OURCH
C168:        109 *
C168:4C 9A CC 110 B.CLREOL JMP X.GS          ;clear to end of line

```



```

C16B:          111 *
C16B:          112 * Clear to end of line using Y = BAS2L
C16B:          113 * which was set up by the $F8 ROM
C16B:          114 *
C16B:A4 2A    115 B.CLREOLZ LDY BAS2L      ;get Y
C16D:4C 9D CC 116          JMP X.GSEOLZ    ;clear to end of line
C170:          117 *
C170:4C 74 CC 118 B.CLREOP JMP X.VT        ;CLEAR TO EOS
C173:4C A0 C2 119 B.SETWND JMP B.SETWNDX
C176:4C B0 C2 120 B.RESET JMP B.RESETX      ;MUST BE IN BFUNC PAGE
C179:4C F2 C2 121 B.RDKEY JMP B.RDKEYX
C17C:          122 *
C17C:20 90 CC 123 B.HOME JSR X.FF          ;HOME & CLEAR
C17F:AD 7B 05 124          LDA OURCH
C182:85 24    125          STA CH          ;COPY CH/CV FOR CALLER
C184:8D 7B 04 126          STA OLDCH       ;REMEMBER WHAT WE SET
C187:4C FE CD 127          JMP VTAB        ;calc base & return
C18A:          128 *
C18A:          129 * Complete PR# or IN# call. Quit video firmware
C18A:          130 * if PR#0 and it was active (B.QUIT). Complete call
C18A:          131 * if inactive (F.QUIT).
C18A:          132 *
C18A:          133 B.QUIT EQU *
C18A:B4 00    134          LDY LOCO,X      ;was it PR#0/IN#0?
C18C:F0 0F    135          BEQ NOTO        ;=>no, not slot 0
C18E:C0 1B    136          CPY #KEYIN      ;was it IN#0?
C190:F0 0E    137          BEQ ISO         ;=>yes, update high byte
C192:20 80 CD 138          JSR QUIT        ;quit the firmware
C195:B4 00    139 F.QUIT LDY LOCO,X      ;get low byte into Y
C197:F0 04    140          BEQ NOTO        ;not slot 0, firmware inactive
C199:A9 FD    141 F8HOOK LDA #<KEYIN      ;set high byte to $FD
C19B:95 01    142          STA LOCL,X
C19D:B5 01    143 NOTO LDA LOCL,X      ;restore accumulator
C19F:60        144          RTS
C1A0:          145 *
C1A0:A5 37    146 ISO LDA CSWH          ;is $C3 in output hook?
C1A2:C9 C3    147          CMP #<BASICIN
C1A4:D0 F3    148          BNE F8HOOK          ;=>no, set to $FDOC
C1A6:4C 32 C8 149          JMP C3IN        ;else set to $C305, exit A=$C3
C1A9:          150 *
C1A9:A4 24    151 F.RDKEY LDY CH          ;else do normal 40 cursor
C1AB:B1 28    152          LDA (BASL),Y      ;grab the character
C1AD:48        153          PHA
C1AE:29 3F    154          AND #$3F        ;set screen to flash
C1B0:09 40    155          ORA #$40
C1B2:91 28    156          STA (BASL),Y    ;and display it
C1B4:68        157 F.NOCUR PLA
C1B5:60        158          RTS          ;return (A=char)
C1B6:          159 *
C1B6:A8        160 F.BASCALC TAY          ;restore Y
C1B7:A5 28    161          LDA BASL        ;restore A
C1B9:20 BA CA 162          JSR BASCALC    ;calculate base address
C1BC:90 4C    163          BCC F.RETURN    ;BASCALC always returns BCC!
C1BE:          164 *

```

```

C1BE:      C1BE 165 B.ESCFIX EQU *
C1BE:20 14 CE 166      JSR UPSHFT      ;upshift lowercase
C1C1:A0 03    167 B.ESCFIX1 LDY #4-1    ;SCAN FOR A MATCH
C1C3:      C1C3 168 B.ESCFIX2 EQU *
C1C3:D9 EE C2 169      CMP ESCIN,Y      ;IS IT?
C1C6:D0 03    C1CB 170      BNE B.ESCFIX3 ;=>NAW
C1C8:B9 A4 C9 171      LDA ESCOUT,Y      ;YES, TRANSLATE IT
C1CB:      C1CB 172 B.ESCFIX3 EQU *
C1CB:88      173      DEY
C1CC:10 F5    C1C3 174      BPL B.ESCFIX2
C1CE:30 3A    C2DA 175      BMI F.RETURN  ;RETURN:CHAR IN AC
C1D0:      176 *
C1D0:20 70 C8 177 F.BOUT JSR BOUT      ;print the character
C1D3:4C 0A C2 178      JMP F.RETURN  ;AND RETURN
C1D6:      179 *
C1D6:      180 * Do displaced mnemonic stuff
C1D6:      181 *
C1D6:8A      182 MNNDX TXA          ;get old acc
C1D7:29 03    183      AND #$03        ;make it a length
C1D9:85 2F    184      STA LENGTH
C1DB:A5 2A    185      LDA BAS2L      ;get old Y into A
C1DD:29 8F    186      AND #$8F
C1DF:4C 71 CA 187      JMP DOMN      ;and go to open spaces
C1E2:      188 *
C1E2:20 F0 FC 189 GOMINI JSR MINI      ;do mini-assembler
C1E5:8A      190      TXA          ;X=0. Set mode to 0, and counter
C1E6:85 34    191      STA YSAV      ;so not CR on new line
C1E8:60      192      RTS
C1E9:      193 *
C1E9:      194 * Pick an 80 column character for the monitor
C1E9:      195 *
C1E9:AC 7B 05 196 FIXPICK LDY OURCH      ;get 80 column cursor
C1EC:20 44 CE 197      JSR PICK      ;pick the character
C1EF:09 80    198      ORA #$80      ;always pick as normal
C1F1:60      199      RTS          ;and return
C1F2:      200 *
C1F2:      201 * Load CH into Y and clear line
C1F2:      202 *
C1F2:      C1F2 203 F.CLREOL EQU *
C1F2:A4 24    204 X.CLREOL LDY CH      ;get horizontal position
C1F4:A9 A0    205 X.CLREOLZ LDA #$A0      ;store a normal blank
C1F6:2C 1E C0 206      BIT ALTCHARSET ;unless alternate char set
C1F9:10 06    C2D1 207      BPL X.CLREOL2
C1FB:24 32    208      BIT INVFLG      ;and inverse
C1FD:30 02    C2D1 209      BMI X.CLREOL2
C1FF:A9 20    210      LDA #$20      ;use inverse blank
C201:4C A8 CC 211 X.CLREOL2 JMP CLR40    ;clear to end of line
C204:      212 *
C204:      213 * Call VTAB or VTABZ for 40 or 80 columns. Acc (CV)
C204:      214 * is saved in BASL.
C204:      215 *
C204:A8      216 F.VTABZ TAY          ;restore Y
C205:A5 28    217      LDA BASL      ;and A
C207:20 03 CE 218      JSR VTABZ      ;do VTABZ

```

```

C20A:          219 *
C20A:          220 * EXIT. EITHER EXIT WITH OR WITHOUT
C20A:          221 * ENABLING I/O SPACE.
C20A:          222 *
C20A:      C20A 223 F.RETURN EQU *
C20A:28        224          PLP          ;GET PRIOR I/O DISABLE
C20B:30 03    C210 225 F.RET2 BMI F.RET1  ;=>LEAVE IT DISABLED
C20D:4C C5 FE 226          JMP FUNCEXIT  ;=>EXIT & ENABLE I/O
C210:4C C8 FE 227 F.RET1 JMP FUNCEXIT+3 ;EXIT DISABLED
C213:          228 *
C213:          229 * Do BOUT, ESCFIX, BASCALC, and KEYIN immediately
C213:          230 * to avoid destroying Accumulator.
C213:          231 *
C213:88        232 DISPATCH DEY
C214:30 BA    C1D0 233          BMI F.BOUT   ;code 0 = 80 column output
C216:88        234          DEY
C217:30 A5    C1BE 235          BMI B.ESCFIX  ;code 1 = ESCFIX
C219:88        236          DEY
C21A:30 9A    C1B6 237          BMI F.BASCALC ;code 2 = BASCALC
C21C:88        238          DEY
C21D:30 3D    C25C 239          BMI B.KEYIN   ;code 3 = KEYIN
C21F:88        240          DEY
C220:30 E2    C204 241          BMI F.VTABZ   ;code 4 = VTABZ
C222:          242 *
C222:          243 * First push address of generic return routine
C222:          244 *
C222:A9 C2     245          LDA #<F.RETURN ;return to F.RETURN
C224:48        246          PHA
C225:A9 09     247          LDA #>F.RETURN-1
C227:48        248          PHA
C228:          249 *
C228:          250 * If any of 5 bits in $4FB (MODE) is on, then the mode is not
C228:          251 * valid for video firmware. Use old routines.
C228:          252 *
C228:AD FB 04 253          LDA MODE       ;no, is mode valid?
C22B:29 D6     254          AND #M.PASCAL+M.6+M.4+M.2+M.1
C22D:D0 0D    C23C 255          BNE GETFUNC  ;=>no, use 40 column routines
C22F:98        256          TYA           ;80 column routines in
C230:18        257          CLC           ;2nd half of table
C231:69 0C     258          ADC #TABLEN
C233:48        259          PHA
C234:20 50 C8 260          JSR CSETUP    ;set up 80 column cursor
C237:20 FE CD 261          JSR VTAB     ;calc base
C23A:68        262          PLA
C23B:A8        263          TAY         ;restore Y
C23C:          264 *
C23C:          265 * Now push address of routine
C23C:          266 *
C23C:A9 C1     267 GETFUNC LDA #<BFUNCPG ;stuff routine address
C23E:48        268          PHA
C23F:B9 44 C2 269          LDA F.TABLE,Y
C242:48        270          PHA
C243:          271 *
C243:          272 * RTS goes to routine on stack. When the routine

```

```

C243:      273 * does an RTS, it returns to F.RETURN, which restores
C243:      274 * the INTCXROM status and returns.
C243:      275 *
C243:60    276      RTS
C244:      277 *
C244:      278 * Table of routines to call. All routines are
C244:      279 * in the $C100 page. These are low bytes only.
C244:      280 *
C244:      C244 281 F.TABLE EQU *
C244:18    282      DFB #>F.HOME-1 ;(5) 40 column HOME
C245:22    283      DFB #>F.SCROLL-1 ;(6) 40 column scroll
C246:F1    284      DFB #>F.CLREOL-1 ;(7) 40 column clear line
C247:5F    285      DFB #>F.CLREOLZ-1 ;(8) 40 column clear with Y set
C248:75
286      DFB #>B.RESET-1 ;(9) 40/80 column reset
C249:02    287      DFB #>F.CLREOP-1 ;(A) 40 column clear end of page
C24A:A8    288      DFB #>F.RDKEY-1 ;(B) readkey w/flashing checkerboard
C24B:51    289      DFB #>F.SETWND-1 ;(C) Set 40 column window
C24C:E1    290      DFB #>GOMINI-1 ;(D) Mini-assembler
C24D:94    291      DFB #>F.QUIT-1 ;(E) quit before IN#0,PR#0
C24E:E8    292      DFB #>FIXPICK-1 ;(F) fix pick for 80 columns
C24F:D5    293      DFB #>MNNDX-1 ;(10) calc mnemonic index
C250:      294 *
C250:      000C 295 TABLEN EQU *-F.TABLE
C250:      296 *
C250:7B    297      DFB #>B.HOME-1 ;(11) 80 column HOME
C251:64    298      DFB #>B.SCROLL-1 ;(12) 80 column scroll
C252:67    299      DFB #>B.CLREOL-1 ;(13) 80 column clear line
C253:6A    300      DFB #>B.CLREOLZ-1 ;(14) 80 column clear with Y set
C254:75    301      DFB #>B.RESET-1 ;(15) 40/80 column reset
C255:6F    302      DFB #>B.CLREOP-1 ;(16) 80 column clear end of page
C256:78    303      DFB #>B.RDKEY-1 ;(17) readkey w/inverse cursor
C257:72    304      DFB #>B.SETWND-1 ;(18) 40/80 column VTAB
C258:E1    305      DFB #>GOMINI-1 ;(19) Mini-Assembler
C259:89    306      DFB #>B.QUIT-1 ;(1A) quit before IN#0,PR#0
C25A:E8    307      DFB #>FIXPICK-1 ;(1B) fix pick for 80 columns
C25B:D5    308      DFB #>MNNDX-1 ;(1C) calc mnemonic index
C25C:      309 *
C25C:      C25C 310 B.KEYIN EQU *
C25C:2C 1F C0 311      BIT RD8OVID ;80 columns?
C25F:10 06 C267 312      BPL B.KEYINI ;=>no, flash the cursor
C261:20 74 C8 313      JSR BIN ;get a keystroke
C264:4C 0A C2 314      GOF.RET JMP F.RETURN ;and return
C267:      315 *
C267:A8    316      B.KEYINI TAY ;preserve A
C268:8A    317      TXA ;put X on stack
C269:48    318      PHA
C26A:98    319      TYA ;restore A
C26B:48    320      PHA ;save char on stack
C26C:48    321      PHA ;dummy for cursor/char test
C26D:      322 *
C26D:68    323      NEW.CUR PLA ;get last cursor
C26E:C9 FF 324      CMP #$FF ;was it checkerboard?
C270:F0 04 C27 6 325      BEQ NEW.CUR1 ;=>yes, get old char

```

```

C272:A9 FF      326      LDA  #$FF      ;no, get checkerboard
C274:D0 02      C278    327      BNE  NEW.CUR2    ;=>always
C276:68         328    NEW.CUR1 PLA      ;get character
C277:48         329      PHA      ;into accumulator
C278:48         330    NEW.CUR2 PHA     ;save for next cursor check
C279:A4 24      331      LDY  CH      ;get cursor horizontal
C27B:91 28      332      STA  (BASL),Y ;and save char/cursor
C27D:           333 *
C27D:           334 * Now leave char/cursor for awhile or
C27D:           335 * until a key is pressed.
C27D:           336 *
C27D:E6 4E      337    WAITKEY1 INC  RNDL      ;bump random seed
C27F:D0 0A      C28B    338      BNE  WAITKEY4    ;=>and check keypress
C281:A5 4F      339      LDA  RNDH      ;is it time to blink yet?
C283:E6 4F      340      INC  RNDH
C285:45 4F      341      EOR  RNDH
C287:29 40      342      AND  #$40
C289:D0 E2      C26D    343      BNE  NEW.CUR      ;=>yes, blink it
C28B:AD 00 C0    344    WAITKEY4 LDA  KBD      ;Ivories been tickled?
C28E:10 ED      C27D    345      BPL  WAITKEY1    ;no, keep blinking
C290:           346 *
C290:68         347      PLA      ;pop char/cursor
C291:68         348      PLA      ;pop character
C292:A4 24      349      LDY  CH      ;and display it
C294:91 28      350      STA  (BASL),Y ;(erase cursor)
C296:68         351      PLA      ;restore X
C297:AA         352      TAX
C298:AD 00 C0    353      LDA  KBD      ;now retrieve the key
C29B:8D 10 C0    354      STA  KBDSTRB ;clear the strobe
C29E:30 C4      C264    355      BMI  GOF.RET    ;=>exit always
C2A0:           356 *
C2A0:           C2A0    357    B.SETWDX EQU *
C2A0:20 52 C1    358      JSR  F.SETWDX ;set 40 column width
C2A3:2C 1F C0    359      BIT  RD80VID ;80 columns?
C2A6:10 02      C2AA    360      BPL  SKPSHFT    ;=>no, width ok
C2A8:06 21      361      ASL  WNDWDTH ;make it 80
C2AA:A5 25      362    SKPSHFT LDA  CV
C2AC:8D FB 05    363      STA  OURCV      ;update OURCV
C2AF:60         364      RTS
C2B0:           365 *
C2B0:           366 * HANDLE RESET FOR MONITOR:
C2B0:           367 *
C2B0:           C2B0    368    B.RESETX EQU *
C2B0:A9 FF      369      LDA  #$FF      ;DESTROY MODE BYTE
C2B2:8D FB 04    370      STA  MODE
C2B5:AD 5D C0    371      LDA  CLRAN2    ;SETUP
C2B8:AD 5F C0    372      LDA  CLRAN3    ; ANNUNCIATORS
C2BB:           373 *
C2BB:           374 * IF THE OPEN APPLE KEY
C2BB:           375 * (ALIAS PADDLE BUTTONS 0) IS
C2BB:           376 * DEPRESSED, COLDSTART THE SYSTEM
C2BB:           377 * AFTER DESTROYING MEMORY:
C2BB:           378 *
C2BB:AD 62 C0    379      LDA  BUTN1      ;GET BUTTON 1 (SOLID)

```

```

C2BE:10 03 C2C3 380 BPL NODIAGS ;=>Up, no diag
C2C0:4C 00 C6 381 JMP DIAGS ;=>else go do diagnostics
C2C3:AD 61 C0 382 NODIAGS LDA BUTNO ;GET BUTTON 0 (OPEN)
C2C6:10 1A C2E2 383 BPL RESETRET ;=>NOT JIVE OR DIAGS
C2C8: 384 *
C2C8: 385 * BLAST 2 BYTES OF EACH PAGE,
C2C8: 386 * INCLUDING THE RESET VECTOR:
C2C8: 387 *
C2C8:A0 B0 388 LDY #$B0 ;LET IT PRECESS DOWN
C2CA:A9 00 389 LDA #0
C2CC:85 3C 390 STA A1L
C2CE:A9 BF 391 LDA #$BF ;START FROM BFXX DOWN
C2D0:38 392 SEC ;FOR SUBTRACT
C2D1: C2D1 393 BLAST EQU *
C2D1:85 3D 394 STA A1H
C2D3:48 395 PHA ;save acc to store
C2D4:A9 A0 396 LDA #$A0 ;blanks
C2D6:91 3C 397 STA (A1L),Y
C2D8:88 398 DEY
C2D9:91 3C 399 STA (A1L),Y
C2DB:68 400 PLA ;restore acc for counter
C2DC:E9 01 401 SBC #1 ;BACK DOWN TO NEXT PAGE
C2DE:C9 01 402 CMP #1 ;STAY AWAY FROM STACK!
C2E0:D0 EF C2D1 403 BNE BLAST
C2E2: 404 *
C2E2: 405 * If there is a ROM card plugged into slot 3,
C2E2: 406 * don't switch in the internal ROM C3 space. If not,
C2E2: 407 * only switch them in if there is a RAM card
C2E2: 408 * in the video slot.
C2E2: 409 *
C2E2: 410 * NOTE: The //e powers up with internal $C3 ROM switched
C2E2: 411 * in. TSTROMCARD switches it out, RESETRET may or may
C2E2: 412 * not switch it back in.
C2E2: 413 *
C2E2: C2E2 414 RESETRET EQU *
C2E2:8D 0B C0 415 STA SETSLOT3ROM ;swap in slot 3
C2E5:20 89 CA 416 JSR TSTROMCRD ;ROM or no card plugged in?
C2E8:D0 03 C2ED 417 BNE GORETN1 ;=>ROM or no card, leave $C3 slot
C2EA:8D 0A C0 418 STA SETINTC3ROM ;card, enable internal ROM
C2ED:60 419 GORETN1 RTS
C2EE: 420 *
C2EE:88 95 8A 8B 421 ESCIN DFB $88,$95,$8A,$8B
C2F2: 422 *
C2F2:A4 24 423 B.RDKEYX LDY CH ;get cursor position
C2F4:B1 28 424 LDA (BASL),Y ;and character
C2F6:2C 1F C0 425 BIT RD80VID ;80 columns?
C2F9:30 F2 C2ED 426 BMI GORETN1 ;=>don't display cursor
C2FB:4C 26 CE 427 JMP INVERT ;else display cursor, exit
C2FE: 428 *
C2FE: 0002 429 ZSPAREC2 EQU C3ORG-*
C2FE: 0002 430 DS C3ORG-*,0
C300: 0000 431 IFNE *-C3ORG
S 432 FAIL 2,'C300 overflow'
C300: 433 FIN

```

```

C300:      19      INCLUDE C3SPACE
C300:      1 *****
C300:      2 *
C300:      3 * THIS IS THE $C3XX ROM SPACE:
C300:      4 * Note: This page must not be used by any routines
C300:      5 * called by the F8 ROM. When it is referenced, it claims
C300:      6 * the C800 space (kicking out anyone who was using it).
C300:      7 * This also means that peripheral cards cannot use the AUXMOVE
C300:      8 * and XFER routines from their C800 space.
C300:      9 *
C300:     10 *****
C300:      C300 11 CNOO EQU *
C300:      C300 12 BASICINT EQU *
C300:2C 43 CE 13 BIT SEV ;set vflag (init)
C303:70 12 C317 14 BVS BASICENT ;(ALWAYS TAKEN)
C305:      15 *
C305:      16 * BASIC input entry point. After a PR#3, this is the
C305:      17 * address that is called to input each character.
C305:      18 *
C305:      C305 19 BASICIN EQU *
C305:38      20 SEC
C306:90      21 DFB $90 ;BCC OPCODE (NEVER TAKEN)
C307:      22 *
C307:      23 * BASIC output entry point. After a PR#3, this is the
C307:      24 * address that is called to output each character.
C307:      25 *
C307:      C307 26 BASICOUT EQU *
C307:18      27 CLC
C308:B8      28 CLV ;CLEAR VFLAG (NOT INIT)
C309:50 0C C317 29 BVC BASICENT ;(ALWAYS TAKEN)
C30B:      30 *
C30B:      31 * Pascal 1.1 Firmware Protocol table:
C30B:      32 *
C30B:      33 * This tables identifies this as an Apple //e 80 column
C30B:      34 * card. It points to the four routines available to
C30B:      35 * programs doing I/O using the Pascal 1.1 Firmware
C30B:      36 * Protocol.
C30B:      37 *
C30B:01      38 DFB $01 ;GENERIC SIGNATURE BYTE
C30C:88      39 DFB $88 ;DEVICE SIGNATURE BYTE
C30D:      40 *
C30D:4A      41 DFB #>JPINIT ;PASCAL INIT
C30E:50      42 DFB #>JPREAD ;PASCAL READ
C30F:56      43 DFB #>JPWRITE ;PASCAL WRITE
C310:5C      44 DFB #>JPSTAT ;PASCAL STATUS
C311:      45 *****
C311:      46 *
C311:      47 * 128K SUPPORT ROUTINE ENTRIES:
C311:      48 *
C311:4C 76 C3 49 JMP MOVE ;MEMORY MOVE ACROSS BANKS
C314:4C C3 C3 50 JMP XFER ;TRANSFER ACROSS BANKS
C317:      51 *****
C317:      52 *
C317:8D 7B 06 53 BASICENT STA CHAR

```



```

C31A:98      54      TYA          ; AND Y
C31B:48      55      PHA
C31C:8A      56      TXA          ; AND X
C31D:48      57      PHA
C31E:08      58      PHP          ;SAVE CARRY & VFLAG
C31F:        59      *
C31F:        60      * If escape mode is allowed, the high bit of MSLOT is
C31F:        61      * clear. Set M.CTL to flag that 1) escapes are allowed, and
C31F:        62      * 2) that control characters should not be echoed.
C31F:        63      * M.CTL is cleared by BPRINT.
C31F:        64      *
C31F:AD FB 04 65      LDA  MODE      ;else esc enable, ctl disable
C322:2C F8 07 66      BIT  MSLOT      ;get MSLOT
C325:30 05 C32C 67      BMI  NOGETLN    ;=>Esc disable, ctl char enable
C327:09 08      68      ORA  #M.CTL
C329:8D FB 04 69      STA  MODE
C32C:        70      *
C32C:        71      NOGETLN EQU  *
C32C:20 6D C3 72      JSR  SETC8      ;SETUP C8 INDICATOR
C32F:28      73      PLP          ;GET VFLAG (INIT)
C330:70 15 C347 74      BVS  JBASINIT  ;=>DO THE INIT
C332:        75      *
C332:        76      * If a PR#0 has been done, input should be transferred
C332:        77      * from the video firmware to KEYIN. This is detected
C332:        78      * if the high bit of the mode byte is set.
C332:        79      *
C332:90 10 C344 80      BCC  JC8        ;=>output , no problem
C334:AA      81      TAX          ;test mode
C335:10 0D C344 82      BPL  JC8        ;video firmware is on
C337:20 5B CD 83      JSR  SETKEYIN  ;else set FDI B as input
C33A:68      84      PLA          ;restore registers
C33B:AA      85      TAX
C33C:68      86      PLA
C33D:A8      87      TAY
C33E:AD 7B 06 88      LDA  CHAR
C341:6C 38 00 89      JMP  (KSWL)    ;go input the character
C344:        90      *
C344:4C 7C C8 91      JC8      JMP  C8BASIC  ;GET OUT OF CN SPACE
C347:4C 03 C8 92      JBASINIT JMP BASICINIT ;=>GOTO C8 SPACE
C34A:        93      *
C34A:        94      JPINIT EQU  *
C34A:20 6D C3 95      JSR  SETC8      ;SETUP C8 INDICATOR
C34D:4C B4 C9 96      JMP  PINIT      ;XFER TO PASCAL INIT
C350:        97      JPREAD EQU  *
C350:20 6D C3 98      JSR  SETC8      ;SETUP C8 INDICATOR
C353:4C D6 C9 99      JMP  PREAD      ;XFER TO PASCAL READ
C356:        100     JPWRITE EQU  *
C356:20 6D C3 101     JSR  SETC8      ;SETUP C8 INDICATOR
C359:4C F0 C9 102     JMP  PWRITE      ;XFER TO PASCAL WRITE
C35C:        103     *
C35C:AA      104     JPSTAT TAX          ;is request code = 0?
C35D:F0 08 C367 105     BEQ  PIORDY    ;=>yes, ready for output
C35F:CA      106     DEX          ;check for any input
C360:D0 07 C369 107     BNE  PSTERR    ;=>bad request, return error

```



```

C362:2C 00 C0      108          BIT   KBD           ;look for a key
C365:10 04 C36B    109          BPL   PNOTRDY        ;=>no keystroked
C367:38           110          PIORDY  SEC
C368:60           111          RTS
C369:           112          *
C369:A2 03         113          PSTERR LDX #3           ;else flag error
C36B:18           114          PNOTRDY CLC
C36C:60           115          RTS
C36D:           116          *****
C36D:           117          * NAME      : SETC8
C36D:           118          * FUNCTION: SETUP IRQ $C800 PROTOCOL
C36D:           119          * INPUT   : NONE
C36D:           120          * OUTPUT  : NONE
C36D:           121          * VOLATILE: NOTHING
C36D:           122          * CALLS   : NOTHING
C36D:           123          *****
C36D:           124          *
C36D:           C36D 125          SETC8 EQU *
C36D:A2 C3         126          LDX   #<CNOO          ;SLOT NUMBER
C36F:8E F8 07      127          STX   MSLOT          ;STUFF IT
C372:AE FF CF      128          LDX   $CFFF          ;kick out other $C8 ROMs
C375:60           129          RTS
C376:           130          *****
C376:           131          * NAME      : MOVE
C376:           132          * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
C376:           133          * INPUT   : A1=SOURCE ADDRESS
C376:           134          *          : A2=SOURCE END
C376:           135          *          : A4=DESTINATION START
C376:           136          *          : CARRY SET=MAIN-->CARD
C376:           137          *          CLR=CARD-->MAIN
C376:           138          * OUTPUT  : NONE
C376:           139          * VOLATILE: NOTHING
C376:           140          * CALLS   : NOTHING
C376:           141          *****
C376:           142          *
C376:           C376 143          MOVE EQU *
C376:48           144          PHA                   ;SAVE AC
C377:98           145          TYA                   ; AND Y
C378:48           146          PHA
C379:AD 13 C0      147          LDA   RDRAMRD        ;SAVE STATE OF
C37C:48           148          PHA                   ; MEMORY FLAGS
C37D:AD 14 C0      149          LDA   RDRAMWRT
C380:48           150          PHA
C381:           151          *
C381:           152          * SET FLAGS FOR CROSSBANK MOVE:
C381:           153          *
C381:90 08 C38B    154          BCC   MOVEC2M        ;=>CARD-->MAIN
C383:8D 02 C0      155          STA   RDMAINRAM      ;SET FOR MAIN
C386:8D 05 C0      156          STA   WRCARDRAM      ; TO CARD
C389:B0 06 C391    157          BCS   MOVESTRT        ;=>(ALWAYS TAKEN)
C38B:           158          *
C38B:           C38B 159          MOVEC2M EQU *
C38B:8D 04 C0      160          STA   WRMAINRAM      ;SET FOR CARD
C38E:8D 03 C0      161          STA   RDCARDRAM      ; TO MAIN

```

```

C391:          162 *
C391:      C391 163 MOVESTRT EQU *
C391:A0 00      164          LDY #0          ;DUMMY INDEX
C393:          165 *
C393:      C393 166 MOVELOOP EQU *
C393:B1 3C      167          LDA (A1L),Y      ;GET A BYTE
C395:91 42      168          STA (A4L),Y      ;MOVE IT
C397:E6 42      169          INC A4L
C399:D0 02      C39D 170          BNE NXTA1
C39B:E6 43      171          INC A4H
C39D:A5 3C      172 NXTA1  LDA A1L
C39F:C5 3E      173          CMP A2L
C3A1:A5 3D      174          LDA A1H
C3A3:E5 3F      175          SBC A2H
C3A5:E6 3C      176          INC A1L
C3A7:D0 02      C3AB 177          BNE C01
C3A9:E6 3D      178          INC A1H
C3AB:90 E6      C393 179 C01      BCC MOVELOOP ;=>MORE TO MOVE
C3AD:          180 *
C3AD:          181 * RESTORE ORIGINAL FLAGS:
C3AD:          182 *
C3AD:8D 04 C0   183          STA WRMAINRAM ;CLEAR FLAG2
C3B0:68         184          PLA          ;GET ORIGINAL STATE
C3B1:10 03      C3B6 185          BPL C03      ;=>IT WAS OFF
C3B3:8D 05 C0   186          STA WRCARDRAM
C3B6:          C3B6 187 C03      EQU *
C3B6:8D 02 C0   188          STA RDMAINRAM ;CLEAR FLAG1
C3B9:68         189          PLA          ;GET ORIGINAL STATE
C3BA:10 03      C3BF 190          BPL MOVERET ;=>IT WAS OFF
C3BC:8D 03 C0   191          STA RDCARDRAM
C3BF:          C3BF 192 MOVERET EQU *
C3BF:68         193          PLA          ;RESTORE Y
C3C0:A8         194          TAY
C3C1:68         195          PLA          ; AND AC
C3C2:60         196          RTS
C3C3:          197 *****
C3C3:          198 * NAME      : XFER
C3C3:          199 * FUNCTION: TRANSFER CONTROL CROSSBANK
C3C3:          200 * INPUT   : $03ED=TRANSFER ADDR
C3C3:          201 *          : CARRY SET=XFER TO CARD
C3C3:          202 *          : CLR=XFER TO MAIN
C3C3:          203 *          : VFLAG CLR=USE STD ZP/STK
C3C3:          204 *          : SET=USE ALT ZP/STK
C3C3:          205 * OUTPUT  : NONE
C3C3:          206 * VOLATILE: $03ED/03EE IN DEST BANK
C3C3:          207 * CALLS   : NOTHING
C3C3:          208 * NOTE    : ENTERED VIA JMP, NOT JSR
C3C3:          209 *****
C3C3:          210 *
C3C3:      C3C3 211 XFER      EQU *
C3C3:48         212          PHA          ;SAVE AC ON CURRENT STACK
C3C4:          213 *
C3C4:          214 * COPY DESTINATION ADDRESS TO THE
C3C4:          215 * OTHER BANK SO THAT WE HAVE IT

```

```

C3C4:          216 *   IN CASE WE DO A SWAP:
C3C4:          217 *
C3C4:AD ED 03  218         LDA  $03ED      ;GET XFERADDR LO
C3C7:48        219         PHA           ;SAVE ON CURRENT STACK
C3C8:AD EE 03  220         LDA  $03EE      ;GET XFERADDR HI
C3CB:48        221         PHA           ;SAVE IT TOO
C3CC:          222 *
C3CC:          223 * SWITCH TO APPROPRIATE BANK:
C3CC:          224 *
C3CC:90 08    C3D6  225         BCC  XFERC2M    ;=>CARD-->MAIN
C3CE:8D 03 C0    226         STA  RDCARDRAM ;SET FOR RUNNING
C3D1:8D 05 C0    227         STA  WRCARDRAM ; IN CARD RAM
C3D4:B0 06    C3DC  228         BCS  XFERZP    ;=> always taken
C3D6:         C3D6  229 XFERC2M EQU  *
C3D6:8D 02 C0    230         STA  RDMAINRAM ;SET FOR RUNNING
C3D9:8D 04 C0    231         STA  WRMAINRAM ; IN MAIN RAM
C3DC:          232 *
C3DC:         C3DC  233 XFERZP EQU  *          ;SWITCH TO ALT ZP/STK
C3DC:68        234         PLA           ;STUFF XFERADDR
C3DD:8D EE 03    235         STA  $03EE      ; HI AND
C3E0:68        236         PLA
C3E1:8D ED 03    237         STA  $03ED      ; LO
C3E4:68        238         PLA          ;RESTORE AC
C3E5:70 05    C3EC  239         BVS  XFERAZP    ;=>switch in alternate zp
C3E7:8D 08 C0    240         STA  SETSTDZP    ;else force standard zp
C3EA:50 03    C3EF  241         BVC  JMPDEST    ;=>always perform transfer
C3EC:8D 09 C0    242 XFERAZP STA  SETALTZP    ;switch in alternate zp
C3EF:6C ED 03    243 JMPDEST JMP  ($03ED)    ;=>off we go
C3F2:          244 *
C3F2:         0002  245         DS    C3ORG+$F4-*,0 ;pad to interrupt stuff
C3F4:          246 *
C3F4:          247 * This is where the interrupt routine returns to.
C3F4:          248 * At this point the ROM is not necessarily switched in so...
C3F4:          249 *
C3F4:8D 81 C0    250 IRQDONE STA  $C081      ;read ROM, write RAM
C3F7:4C 7A FC    251         JMP  IRQDONE2    ;and jump to ROM
C3FA:          252 *
C3FA:          253 * This is the main entry point for the interrupt
C3FA:          254 * handler. This switches in the internal ROM and
C3FA:          255 * jumps to the main part of the interrupt handler
C3FA:          256 * at $C400.
C3FA:          257 *
C3FA:2C 15 C0    258 irq      bit  rdcxrom    ;Test internal or external rom
C3FD:8D 07 C0    259         sta  setintcxrom ;Force in ROM to get to interrupt handler
C400:          260 *
C400:          261 * Fall into $C400 which is now switched in!!
C400:          262 *
C400:          20      INCLUDE IRQ
C400:          1 *
C400:          2 * Here is the main interrupt handler
C400:          3 *
C400:          4 *****
C400:         C400  5 newirq equ  *
C400:D8        6         cld              ;make no assumptions!!

```

C401:38		7	sec		;C=1 if internal slot space
C402:30 01	C405	8	bmi	irqintcx	
C404:18		9	clc		
C405:48		10	irqintcx	pha	;Save A on stack instead of \$45
C406:48		11		pha	;Make room for rts if needed
C407:48		12		pha	
C408:8A		13		txa	;Save X
C409:BA		14		tsx	;Get stack pointer for BRK bit
C40A:E8		15		inx	;Can't do add cause we need C
C40B:E8		16		inx	
C40C:E8		17		inx	
C40D:E8		18		inx	
C40E:48		19		pha	
C40F:98		20		tya	;and Y
C410:48		21		pha	
C411:BD 00 01		22		lda \$100,x	;Get status for break test
C414:29 10		23		and #\$10	;A = \$10 if break
C416:A8		24		tay	;Save it for later
C417:		25	* Now test & set the state of the machine. Don't alter Y		
C417:AD 18 C0		26		lda rd80col	;Test for 80 store and page 2
C41A:2D 1C C0		27		and rdpag2	
C41D:29 80		28		and #\$80	;Make it 0 or \$80
C41F:F0 05 C426		29		beq irq2	;Branch if no change needed
C421:A9 20		30		lda #\$20	;Set shifted page 2 reset bit
C423:8D 54 C0		31		sta txtpagel	;Set page 1
C426:2A		32	irq2	rol A	;Align bit & shift in slotcx bit
C427:2C 13 C0		33		bit rdramrd	;Are we reading from aux ram?
C42A:10 05 C431		34		bpl irq3	;Branch if main ram read
C42C:8D 02 C0		35		sta rdmainram	;Else, switch main in
C42F:09 20		36		ora #\$20	;and record the event
C431:2C 14 C0		37	irq3	bit rdramwrt	;Do the same for ram write
C434:10 05 C43B		38		bpl irq4	
C436:8D 04 C0		39		sta wrmainram	
C439:09 10		40		ora #\$10	
C43B:	C43B	41	irq4	equ *	
C43B:2C 12 C0		42	irq5	bit rdlcram	;Determine if language card active
C43E:10 0C C44C		43		bpl irq7	
C440:09 0C		44		ora #\$0C	;Sets two bits. Second is redundant
C442:2C 11 C0		45		bit rdlcbnk2	;if INC used to restore.
C445:10 02 C449		46		bpl irq6	;Branch if not page 2 of \$D000
C447:49 06		47		eor #\$06	;Set bits for page 2
C449:8D 81 C0		48	irq6	sta romin	;Enable ROM STA leaves write enable alone
C44C:2C 16 C0		49	irq7	bit rdaltzp	;Last...and very important
C44F:10 0D C45E		50		bpl irq8	;If alternate stack
C451:BA		51		tsx	;store current stack pointer at \$101
C452:8E 01 01		52		stx \$101	
C455:AE 00 01		53		ldx \$100	;Retreive main stack pointer from \$100
C458:9A		54		txs	
C459:8D 08 C0		55		sta setstdzp	
C45C:09 80		56		ora #\$80	;Mark stack switched
C45E:88		57	irq8	dey	;Was it a break?
C45F:30 0C C46D		58		bmi irq9	
C461:85 44		59		sta macstat	;Save state of machine
C463:68		60		pla	;Restore registers

```

C464:A8      61      tay
C465:68      62      pla
C466:AA      63      tax
C467:68      64      pla
C468:68      65      pla          ;A stored where RTS address would go
C469:68      66      pla
C46A:4C 47 FA 67      jmp newbreak ;Go to normal break routine stuff
C46D:48      68 irq9  pha          ;Save state of machine on stack
C46E:AD F8 07 69      lda mslot    ;Save mslot
C471:48      70      pha
C472:A9 C3    71      lda #<irqdone ;Save return irq address
C474:48      72      pha
C475:A9 F4    73      lda #>irqdone ;so when interrupt does RTI
C477:48      74      pha          ;It returns to irqdone
C478:08      75      php          ;Status for user's RTI
C479:4C 74 FC 76      jmp irquser  ;Off to the user
C47C:         77 * The user's RTI returns here
C47C:         78 * BEWARE
C47C:         79 * The rom must be reenabled with a LDA romin
C47C:         80 * This way if the LC was write protected, it still is
C47C:         81 * if it was write enabled, it still is
C47C:         82 * if it was being write enabled ( 2 ldas), it still will be
C47C:         83 * The restore loop uses an INC because some of the switches are read
C47C:         84 * and some are write. It must be an INC abs,x since both the 6502 and
C47C:         85 * the 65C02 do two reads before the write.
C47C:AD 81 C0 86 irqfix lda romin    ;Must be lda!
C47F:68      87      pla          ;Recover machine state
C480:10 07 C489 88      bpl irqdn1   ;Branch if main ZP
C482:8D 09 C0 89      sta setaltzp
C485:AE 01 01 90      ldx $101    ;Get alt stack pointer
C488:9A      91      txs
C489:A0 06    92 irqdn1 ldy #$06    ;Y = index into table of switch addresses
C48B:10 06 C493 93 irqdn2 bpl irqdn3   ;Branch if no change
C48D:BE C1 C4 94      ldx irqtbl,y ;Get soft switch address
C490:FE 00 C0 95      inc $C000,x ;Hit the switch. NO PAGE CROSS!
C493:88      96 irqdn3 dey
C494:30 03 C499 97      bmi irqdn4
C496:0A      98      asl A          ;Get next bit to check
C497:D0 F2 C48B 99      bne irqdn2
C499:0A      100 irqdn4 asl A          ;C = 1 if internal slot space
C49A:0A      101      asl A
C49B:68      102      pla          ;Restore the registers
C49C:A8      103      tay
C49D:BA      104      tsx          ;Save the stack pointer
C49E:A9 40    105      lda #$40    ;RTI opcode
C4A0:48      106      pha
C4A1:A9 C0    107      lda #<setslotcxrom
C4A3:48      108      pha
C4A4:A9 06    109      lda #>setslotcxrom
C4A6:69 00    110      adc #0          ;Add 1 if internal slot space
C4A8:48      111      pha
C4A9:A9 8D    112      lda #$8D    ;STA setslotcxrom
C4AB:48      113      pha

```

```

C4AC:9A      114      txs          ;Restore stack pointer
C4AD:8A      115      txa          ;Make return address on stack point to code on stack
C4AE:69 03   116      adc #3       ;C = 0 from earlier adc
C4B0:AA      117      tax
C4B1:38      118      sec
C4B2:E9 07   119      sbc #7       ;Point to where code starts
C4B4:9D 00 01 120      sta $100,x
C4B7:E8      121      inx
C4B8:A9 01   122      lda #$1
C4BA:9D 00 01 123      sta $100,x
C4BD:68      124      pla
C4BE:AA      125      tax
C4BF:68      126      pla
C4C0:60      127      rts          ;Go to code on stack

```

```

C4C1:83 8B 8B 129 irqtbl dfb >1cbank2,>1cbank1,>1cbank1
C4C4:05 03 55 130      dfb >wrcardram,>rdcardram,>txtpage2
C4C7:          21      INCLUDE DIAGS
----- NEXT OBJECT FILE NAME IS REFLIST.1
C600:          C600    1      ORG C3ORG+$300
C600:          2 * These routines test all 64K RAM, as well as the 64K on an Auxiliary
C600:          3 * memory card (when present). With the exception of the INTCXROM switch
C600:          4 * of the IOU, all combinations of the IOU switches are tested and ver-
C600:          5 * ified. All configurations of the MMU switches are also tested.
C600:          6 *
C600:          7 * In the event of any failure, the diagnostic is halted. A message
C600:          8 * is written to screen memory indicating the source of the failure.
C600:          9 * When RAM fails the message is composed of "RAM ZP" (indicating failure
C600:         10 * detected in the first page of RAM) or "RAM" (meaning the other 63.75K),
C600:         11 * followed by a binary representation of the failing bits set to "1".
C600:         12 * For example, "RAM 0 1 1 0 0 0 0 0" indicates that bits 5 and 6 were
C600:         13 * detected as failing. To represent auxiliary memory, a "*" symbol is
C600:         14 * printed preceeding the message.
C600:         15 *
C600:         16 * When the MMU or IOU fail, the message is simply "MMU" or "IOU".
C600:         17 *
C600:         18 * The test will run continuously for as long as the Open and Closed
C600:         19 * Apple keys remain depressed (or no keyboard is connected) and no
C600:         20 * failures are encountered. The message "System OK" will appear in
C600:         21 * the middle of the screen when a successful cycle has been run and
C600:         22 * either of the Apple keys are no longer depressed. Another cycle
C600:         23 * may be initiated by pressing both Apple keys again while this message
C600:         24 * is on the screen. To exit diagnostics, Control-Reset must be pressed
C600:         25 * without the Apple keys depressed.
C600:         26 *
C600:         C0051 27 TEXT equ $C051
C600:         0009 28 IOUIDX equ $09
C600:         0001 29 MMUIDX equ $01
C600:         05B8 30 SCREEN equ $5B8
C600:         C000 31 IOSPACE equ $C000
C600:         32 *
C600:         C600 33 DIAGS equ *

```

```

C600:8D 50 C0      34      sta $C050
C603:              35 * Test Zero-Page, then all of memory. Report errors when encountered.
C603:              36 * Accumulator can be anything on entry. All registers used, but no sta-ck.
C603:              37 * Addresses between $C000 and $CFFF are mapped to main $D000 bank.
C603:              38 * Auxillary 64K is also tested if present.

C603:A0 04        40 TSTZPG ldy #$4
C605:A2 00        41      ldx #0
C607:18          42 zp1    clc                ;fill zero page with a pattern
C608:79 B4 C7     43      adc ntbl,y
C60B:95 00        44      sta $00,x
C60D:E8          45      inx
C60E:D0 F7 C607   46      bne zp1                ;after all bytes filled,
C610:18          47 zp2    clc                ; ACC has original value again.
C611:79 B4 C7     48      adc ntbl,y                ;so values can be tested
C614:D5 00        49      cmp $00,x
C616:D0 10 C628   50      bne ZPERROR                ;branch if memory failed
C618:E8          51      inx
C619:D0 F5 C610   52      bne zp2                ;loop until all 256 bytes tested
C61B:6A          53      ror a                ;change ACC so location $FF will change
C61C:2C 19 C0     54      bit RDVBLBAR                ; use RDVBLBAR for a little randomness...
C61F:10 02 C623   55      bpl zp3
C621:49 A5        56      eor #$A5
C623:88          57 zp3    dey                ;use a different pattern now
C624:10 E1 C607   58      bpl zp1                ;branch to retest with other value
C626:30 06 C62E   59      bmi TSTMEM                ;branch always

C628:55 00        61 ZPERROR eor $00,x                ;which bits are bad?
C62A:18          62      clc                ;indicate zero page failure
C62B:4C CD C6     63      jmp BADBITS
C62E:          C62E 64 TSTMEM equ *
C62E:86 01        65      stx $01
C630:86 02        66      stx $02
C632:86 03        67      stx $03
C634:A2 04        68      ldx #4                ;do RAM $100-$FFFF five times
C636:86 04        69      stx $04
C638:E6 01        70 mem1   inc $01                ;point to page 1 first
C63A:A8          71 mem2   tay                ;save ACC in Y for now
C63B:8D 83 C0     72      sta $C083                ;anticipate not $C000 range...
C63E:8D 83 C0     73      sta $C083
C641:A5 01        74      lda $01                ;get page address
C643:29 F0        75      and #$F0                ;test for $C0-$CF range
C645:C9 C0        76      cmp #$C0
C647:D0 0C C655   77      bne mem3                ;branch if not...
C649:AD 8B C0     78      lda $C08B
C64C:AD 8B C0     79      lda $C08B                ;select primary $D000 space
C64F:A5 01        80      lda $01
C651:69 0F        81      adc #$F                ;Plus carry += $10
C653:D0 02 C657   82      bne mem4                ;branch always taken
C655:A5 01        83 mem3   lda $01
C657:85 03        84 mem4   sta $03
C659:98          85      tya                ;restore pattern to ACC
C65A:A0 00        86      ldy #$00                ;fill this page with the pattern

```


C65C:18	87	mem5	clc	
C65D:7D B4 C7	88		adc	ntbl,x
C660:91 02	89		sta	(\$02),y
C662:CA	90		dex	;keep x in the range 0-4
C663:10 02 C667	91		bpl	mem6
C665:A2 04	92		ldx	#4
C667:C8	93	mem6	iny	;all 256 filled yet?
C668:D0 F2 C65C	94		bne	mem5 ;branch if not
C66A:E6 01	95		inc	1 ;bump page #
C66C:D0 CC C63A	96		bne	mem2 ;loop through \$0100 to \$FF00
C66E:E6 01	98		inc	\$01 ;point to page 1 again
C670:A8	99	mem7	tay	;save ACC in Y for now
C671:AD 83 C0	100		lda	\$C083 ;anticipate not \$C000 range...
C674:AD 83 C0	101		lda	\$C083
C677:A5 01	102		lda	\$01 ;get page address
C679:29 F0	103		and	#\$F0 ;test for \$C0-\$CF range
C67B:C9 C0	104		cmp	#\$C0
C67D:D0 09 C688	105		bne	mem8 ;branch if not...
C67F:AD 8B C0	106		lda	\$C08B ;select primary \$D000 space
C682:A5 01	107		lda	\$01
C684:69 0F	108		adc	#\$F ;Plus carry += \$10
C686:D0 02 C68A	109		bne	mem9 ;branch always taken
C688:A5 01	110	mem8	lda	\$01
C68A:85 03	111	mem9	sta	\$03
C68C:98	112		tya	;restore pattern to ACC
C68D:A0 00	113		ldy	#\$00 ;fill this page with the pattern
C68F:18	114	memA	clc	
C690:7D B4 C7	115		adc	ntbl,x
C693:51 02	116		eor	(\$02),y
C695:D0 35 C6CC	117		bne	MEMERROR ;if any bits are different, give up!!!
C697:B1 02	118		lda	(\$02),y ;restore correct pattern
C699:CA	119		dex	;keep x in the range 0-4
C69A:10 02 C69E	120		bpl	memB
C69C:A2 04	121		ldx	#4
C69E:C8	122	memB	iny	;all 256 filled yet?
C69F:D0 EE C68F	123		bne	memA ;branch if not
C6A1:E6 01	124		inc	1 ;bump page #
C6A3:D0 CB C670	125		bne	mem7 ;loop through \$0100 to \$FF00
C6A5:6A	126		ror	a ;change ACC for next pass
C6A6:2C 19 C0	127		bit	RDVBLBAR ; use RDVBLBAR for a little randomness...
C6A9:10 02 C6AD	128		bpl	memC
C6AB:49 A5	129		eor	#\$A5
C6AD:C6 04	130	memC	dec	\$04 ;have 5 passes been done yet?
C6AF:10 87 C638	131		bpl	mem1 ;branch if not...
C6B1:AA	133		TAX	;save acc
C6B2:20 8D C9	134		JSR	STAUX ;set aux memory & write \$EE to \$C00,\$800
C6B5:D0 07 C6BE	135		BNE	SWCHTST1 ;=>not 128K
C6B7:0E 00 0C	136		ASL	\$C00 ;shift test byte
C6BA:0A	137		ASL	A
C6BB:CD 00 0C	138		CMP	\$C00 ;check memory


```

C6BE:D0 76 C736 139 SWCHTST1 BNE SWCHTST ;=>not 128K
C6C0:CD 00 08 140 CMP $800 ;look for shadowing
C6C3:F0 71 C736 141 BEQ SWCHTST ;=>not 128K
C6C5:8A 142 txa
C6C6:8D 09 C0 143 STA SETALTZP ;swap in alt zero page
C6C9:4C 03 C6 144 jmp TSTZPG ; and test it!
C6CC:38 145 MEMERROR sec ;indicate main ram failure
C6CD:AA 146 BADBITS tax ;save bit pattern in x for now
C6CE:AD 13 C0 147 lda RDRAMRD ;determine if primary or auxillary RAM
C6D1:B8 148 clv ;with V-FLG
C6D2:10 03 C6D7 149 bpl bbits1 ;branch if primary bank
C6D4:2C B4 C7 150 bit setv
C6D7:A9 A0 151 bbits1 lda #$A0 ;try to clear video screen
C6D9:A0 06 152 ldy #6
C6DB:99 FE BF 153 clrsts sta IOSPACE-2,y
C6DE:99 06 C0 154 sta IOSPACE+6,y
C6E1:88 155 dey
C6E2:88 156 dey
C6E3:D0 F6 C6DB 157 bne clrsts
C6E5:8D 51 C0 158 sta TEXT
C6E8:8D 54 C0 159 sta TXTPAGE1
C6EB:99 00 04 160 clr sta $400,y
C6EE:99 00 05 161 sta $500,y
C6F1:99 00 06 162 sta $600,y
C6F4:99 00 07 163 sta $700,y
C6F7:C8 164 iny
C6F8:D0 F1 C6EB 165 bne clr
C6FA:8A 166 txa ;test for switch test failure
C6FB:F0 27 C724 167 beq BADSWTCH ;branch if it was a switch
C6FD:A0 03 168 ldy #3
C6FF:B0 02 C703 169 bcs badmain ;branch if ZP ok
C701:A0 05 170 ldy #5
C703:A9 AA 171 badmain lda #$AA ;mark aux report with an asterisks
C705:50 03 C70A 172 bvc badprim
C707:8D B0 05 173 sta screen-8
C70A:B9 EA C7 174 badprim lda rmess,y
C70D:99 B1 05 175 sta screen-7,y
C710:88 176 dey
C711:10 F7 C70A 177 bpl badprim ;message is either "RAM" or "RAM ZP"
C713:A0 10 178 ldy #$10 ;print bits
C715:8A 179 bbits2 txa
C716:4A 180 lsr a
C717:AA 181 tax
C718:A9 58 182 lda #$58 ;bits are printed as ascii 0 or 1
C71A:2A 183 rol a
C71B:99 B6 05 184 sta screen-2,y
C71E:88 185 dey
C71F:88 186 dey
C720:D0 F3 C715 187 bne bbits2
C722:F0 FE C722 188 hangx beq hangx ;hang forever and ever
C724:A0 02 189 BADSWTCH ldy #2
C726:B9 F0 C7 190 bswtchl lda smess,y
C729:90 03 C72E 191 bcc bswtch2 ;branch if MMU in error
C72B:B9 F3 C7 192 lda smess+3,y ;else indicate IOU error

```

```

C72E:99 B8 05      193 bswtch2 sta screen,y
C731:88            194      dey
C732:10 F2      C726 195      bpl bswtchl ;print "MMU" or "IOU"
C734:30 FE      C734 196 hangy bmi hangy ;branch forever

C736:A0 01      198 SWCHTST ldy #MMUIDX
C738:A9 7F      199 swtst1 lda #$7F
C73A:6A      200 swtst2 ror a ;set switches of the IOU/MMU to match Accumulator
C73B:BE B9 C7   201      ldx SWTBLO,y
C73E:F0 0F      C74F 202      beq swtst4 ;branch if done setting switches
C740:90 03      C745 203      bcc swtst3 ;branch if setting switch to 0-state
C742:BE C9 C7   204      ldx SWTBL1,y ;else get index to set switch to 1
C745:9D FF BF   205 swtst3 sta IOSPACE-1,x ;set switch
C748:C8      206      iny
C749:DO EF      C73A 207      bne swtst2 ;branch always taken...
C74B:      208 *
C74B:AE 30 C0   209 click ldx $C030
C74E:2A      210      rol a
C74F:88      211 swtst4 dey
C750:BE D9 C7   212      ldx RSWTBL,y ;now verify the settings just made
C753:F0 13      C768 213      beq swtst6 ;branch if done this pass
C755:30 F4      C74B 214      bmi click ;branch if this switch no to be verified.
C757:2A      215      rol a
C758:90 07      C761 216      bcc swtst5
C75A:1E 00 C0   217      asl IOSPACE,x
C75D:90 17      C776 218      bcc swerr
C75F:B0 EE      C74F 219      bcs swtst4 ;branch always
C761:1E 00 C0   220 swtst5 asl IOSPACE,x
C764:B0 10      C776 221      bcs swerr
C766:90 E7      C74F 222      bcc swtst4 ;branch always
C768:      223 *
C768:2A      224 swtst6 rol a ;restore original value
C769:C8      225      iny ; and IOU/MMU index
C76A:38      226      sec
C76B:E9 01      227      sbc #1 ;try next pattern
C76D:B0 CB      C73A 228      bcs swtst2
C76F:88      229      dey ;was MMU just tested?
C770:DO 0B      C77D 230      bne BIGLOOP ;branch if IOU was just tested
C772:A0 09      231      ldy #IOUIDX ;else, go test IOU.
C774:DO C2      C738 232      bne swtst1 ;branch always taken...
C776:      233 *
C776:A2 00      234 swerr ldx #0 ;indicate switch error
C778:C0 0A      235      cpy #IOUIDX+1 ;set carry if IOU was cause
C77A:4C D7 C6   236      jmp bbitsl
C77D:46 80      237 BIGLOOP lsr $80
C77F:DO B5      C736 238      bne SWCHTST
C781:A9 A0      239 blp2 lda #$A0
C783:A0 00      240      ldy #0
C785:99 00 04   241 blp3 sta $400,y ;clear screen for success message
C788:99 00 05   242      sta $500,y
C78B:99 00 06   243      sta $600,y
C78E:99 00 07   244      sta $700,y
C791:C8      245      iny

```

```

C792:D0 F1 C785 246 bne blp3
C794:AD 61 C0 247 blp4 LDA $C061 ;test for both Open and Closed Apple
C797:2D 62 C0 248 AND $C062 ; pressed
C79A:0A 249 asl a ;put result in carry
C79B:E6 FF 250 INC $FF
C79D:A5 FF 251 LDA $FF
C79F:90 03 C7A4 252 bcc dquit
C7A1:4C 00 C6 253 jmp DIAGS
C7A4: 254 *
C7A4:AD 51 C0 255 dquit lda TEXT ;put success message on the screen
C7A7:A0 08 256 ldy #8
C7A9:B9 F6 C7 257 suc2 lda success,y
C7AC:99 B8 05 258 sta SCREEN,y
C7AF:88 259 dey
C7B0:10 F7 C7A9 260 bpl suc2
C7B2:30 E0 C794 261 bmi blp4 ;loop forever
C7B4: 262 *
C7B4: C7B4 263 setv equ *
C7B4:53 43 2B 29 264 ntbl dfb 83,67,43,41,7
C7B9:00 89 31 03 265 swtb10 dfb $00,$89,$31,$03,$05,$09,$0b,$01,$00,$83,$51,$53,$55,$57,$0F, $0D
C7C9:00 81 31 04 266 swtb11 dfb $00,$81,$31,$04,$06,$0A,$0C,$02,$00,$84,$52,$54,$56,$58,$10, $0E
C7D9:00 11 FF 13 267 rswtb1 dfb $00,$11,$FF,$13,$14,$16,$17,$18,$00,$12,$1A,$1B,$1C,$1D,$1E, $1F,$00
C7EA: 268 MSB ON
C7EA:D2 C1 CD A0 269 rmess asc "RAM ZP"
C7FA:CD CD D5 C9 270 smess asc "MMUIOU"

C7F6:D3 F9 F3 F4 272 success asc "System OK"
C7FF: C7FF 273 zzzend equ *
C7FF: 22 INCLUDE C8SPACE
C7FF: 0001 1 DS C8ORG-*,0 ;pad to C800
C800: 2 *
C800: 3 * This entry point is only used by Pascal 1.0
C800: 4 *
C800:4C B0 C9 5 JMP PINIT1.0 ;PASCAL 1.0 INIT
C803: 6 *
C803: 7 * BASIC initialization:
C803: 8 *
C803: 9 * This is called by the $C3 space only after a PR#3 or
C803: 10 * the equivalent (a JSR $C300).
C803: 11 *
C803: 12 * It causes a copy of the $F8 ROM to be placed in the
C803: 13 * language card if the language card is switched in and
C803: 14 * the ID byte doesn't match. It sets up all the
C803: 15 * screenhole variables to support its operation. If the
C803: 16 * 80 column card is detected, it sets things up for 80 column
C803: 17 * operation, else 40 column operation. Then it clears the
C803: 18 * screen and prints the character that was in the accumulator
C803: 19 * upon entry.
C803: 20 *
C803: C803 21 BASICINIT EQU *
C803:20 F4 CE 22 JSR COPYROM ;If LC in, copy F8 to it
C806:20 2A C8 23 JSR C3HOOKS ;out=$C307, in=$C305
C809:20 2E CD 24 JSR D040 ;set full 40-col window

```

```

C80C:A9 01      25      LDA #M.MOUSE ;init with mouse text off
C80E:8D FB 04   26      STA MODE ;Set BASIC video mode
C811:           27 *
C811:           28 * IS THERE A CARD?
C811:           29 *
C811:20 90 CA    30      JSR TESTCARD ;SEE IF CARD PLUGGED IN
C814:D0 08 C81E 31      BNE CLEARIT ;=>IT'S 40
C816:06 21      32      ASL WNDWDTH ;SET 80-COL WINDOW
C818:8D 01 C0    33      STA SET80COL ;ENABLE 80 STORE
C81B:8D 0D C0    34      STA SET80VID ; AND 80 VIDEO
C81E:           35 *
C81E:           36 * HOME & CLEAR:
C81E:           37 *
C81E: C81E      38 CLEARIT EQU *
C81E:8D 0F C0    39      STA SETALTCHAR ;SET NORM/INV LCASE
C821:20 90 CC    40      JSR X.FF ;CLEAR IT
C824:AC 7B 05    41      LDY OURCH ;set up cursor for store
C827:4C 7E C8    42      JMP BPRINT ;always print a character
C82A:           43 *
C82A:A9 07      44 C3HOOKS LDA #>BASICOUT ;set output hook first
C82C:85 36      45      STA CSWL
C82E:A9 C3      46      LDA #<CN00
C830:85 37      47      STA CSWH
C832:           48 *
C832:           49 * C3IN is called by IN#0 if CSWH = #$C3
C832:           50 *
C832:A9 05      51 C3IN LDA #>BASICIN ;set input hook
C834:85 38      52      STA KSWL
C836:A9 C3      53      LDA #<CN00
C838:85 39      54      STA KSWH
C83A:60         55      RTS ;exit with A=$C3 for IN#0 stuff
C83B:           56 *
C83B:E6 4E      57 GETKEY INC RNDL ;BUMP RANDOM SEED
C83D:D0 02 C841 58      BNE GETK2
C83F:E6 4F      59      INC RNDH
C841:AD 00 C0    60 GETK2 LDA KBD ;KEYPRESS?
C844:10 F5 C83B 61      BPL GETKEY ;=>NOPE
C846:8D 10 C0    62      STA KBDSTRB ;CLEAR STROBE
C849:60         63      RTS
C84A:           64 *
C84A:           65 *****
C84A:           66 *
C84A:           67 * PASCAL 1.0 INPUT HOOK:
C84A:           68 *
C84A: 00C3      69      DS C8ORG+$4D-*,0 ;pad to 1.0 hooks
C84D: 0000      70      IFNE *-C8ORG-$4D ;ERR IF WRONG ADDR
S      71      FAIL 2,'C84D HOOK ALIGNMENT'
C84D:           72      FIN
C84D:4C 50 C3    73      JMP JPREAD ;=>GO TO STANDARD READ
C850:           74 *****
C850:           75 *
C850:           76 * CSETUP compensates for everything that the user
C850:           77 * can do to change the cursor status: poke CV, CH,
C850:           78 * OURCH, WNDWDTH. It updates the video firmware's

```

```

C850:          79 * versions of these values for its own use.
C850:          80 * COPY USER'S CURSOR IF IT DIFFERS FROM
C850:          81 * WHAT WE LAST PUT THERE:
C850:          82 *
C850:A5 25     83 CSETUP LDA CV ;set up OURCV
C852:8D FB 05  84 STA OURCV
C855:A4 24     85 LDY CH ;GET IT
C857:CC 7B 04  86 CPY OLDCH ;IS IT THE SAME?
C85A:F0 03 C85F 87 BEQ CS2 ;=>YES, USE OUR OWN
C85C:8C 7B 05  88 STY OURCH ;update our cursor
C85F:A5 21     89 CS2 LDA WNDWDTH ;cursor horizontal must not
C861:18        90 CLC ;be greater than window width
C862:ED 7B 05  91 SBC OURCH ;if it is, then put cursor
C865:B0 05 C86C 92 BCS CS3 ;at left edge of window
C867:A0 00     93 LDY #0
C869:8C 7B 05  94 STY OURCH
C86C:AC 7B 05  95 CS3 LDY OURCH ;exit with Y = CH
C86F:60        96 RTS
C870:          97 *
C870:          98 * BIN and BOUT are used when characters are
C870:          99 * input and output by the $F8 ROM while 80VID
C870:         100 * is on. They cannot use the $C3 entry points
C870:         101 * because that switches in the $C8 space, causing
C870:         102 * possible conflict with other $C8 users.
C870:         103 * These routines are only called by the $C100-$C2FF space.
C870:         104 *
C870:         105 * These entry points will only work if the card was
C870:         106 * first initialized using a PR#3. 80 columns will not
C870:         107 * work simply by turning on the 80VID flag.
C870:         108 *
C870:A4 35     109 BOUT LDY SAVY1 ;load Y stuffed by $F8 ROM
C872:18        110 CLC ;signal an output
C873:B0 FE C873 111 BCS * ;skip SEC
C874: C874     112 ORG *-1
C874:38        113 BIN SEC ;signal an input
C875:8D 7B 06  114 STA CHAR ;save the char
C878:98        115 TYA ;save Y
C879:48        116 PHA
C87A:8A        117 TXA ;save X
C87B:48        118 PHA
C87C: C87C     119 C8BASIC EQU * ;BASIC IN/OUT
C87C:B0 5E C8DC 120 BCS BINPUT ;=>input a character
0000: 0000     1 TEST EQU 0 ;REAL VERSION
C87E:          23 LST ON,A,V
C87E:          24 INCLUDE BPRINT
C87E:          1 *
C87E:          2 * This is the place where characters printed using the
C87E:          3 * CSW hook are actually printed (or executed if they are
C87E:          4 * control characters).
C87E:          5 *
C87E:20 50 C8   6 BPRINT JSR CSETUP ;setup user cursor
C881:AD 7B 06   7 LDA CHAR ;GET CHARACTER
C884:C9 8D     8 CMP #$8D ;IS IT C/R?
C886:D0 18 C8A0 9 BNE NOWAIT ;=>don't wait, OURCH ok

```

```

C888:AE 00 C0      10      LDX KBD      ;IS KEY PRESSED?
C88B:10 13 C8A0    11      BPL NOWAIT   ;NO
C88D:E0 93        12      CPX #$93      ;IS IT CTL-S?
C88F:D0 0F C8A0    13      BNE NOWAIT   ;NO, IGNORE IT
C891:2C 10 C0      14      BIT KBDSTRB  ;CLEAR STROBE
C894:AE 00 C0      15 KBDWAIT LDX KBD      ;WAIT FOR NEXT KEYPRESS
C897:10 FB C894    16      BPL KBDWAIT
C899:E0 83        17      CPX #$83      ;IF CTL-C, LEAVE IT
C89B:FO 03 C8A0    18      BEQ NOWAIT   ; IN THE KBD BUFFER
C89D:2C 10 C0      19      BIT KBDSTRB  ;CLEAR OTHER CHARACTER
C8A0:29 7F        20 NOWAIT AND #$7F      ;drop possible hi bit
C8A2:C9 20        21      CMP #$20      ;IS IT CONTROL CHAR?
C8A4:B0 06 C8AC    22      BCS BPNCTL   ;=>NOPE
C8A6:20 D2 CA      23      JSR CTLCHAR0 ;execute CTL if M.CTL ok
C8A9:4C BD C8      24      JMP CTLON    ;=>enable ctl chrs
C8AC:              25 *
C8AC:              26 * NOT A CTL CHAR. PRINT IT.
C8AC:              27 *
C8AC:              28 BPNCTL EQU *
C8AC:AD 7B 06      29      LDA CHAR      ;get char (all 8 bits)
C8AF:20 38 CE      30      JSR STORCHAR ;and display it
C8B2:              31 *
C8B2:              32 * BUMP THE CURSOR HORIZONTAL:
C8B2:              33 *
C8B2:C8           34      INY          ;bump it
C8B3:8C 7B 05      35      STY OURCH      ;are we past the
C8B6:C4 21         36      CPY WNDWDTH    ; end of the line?
C8B8:90 03 C8BD    37      BCC CTLON    ;=>NO, NO PROBLEM
C8BA:20 51 CB      38      JSR X.CR      ;YES, DO C/R
C8BD:              39 *
C8BD:              40 * M.CTL is set by RDCHAR and cleared here, after each
C8BD:              41 * character is displayed.
C8BD:              42 *
C8BD:AD FB 04      43 CTLON LDA MODE      ;enable printing of control chars
C8C0:29 F7         44      AND #255-M.CTL
C8C2:8D FB 04      45      STA MODE
C8C5:AD 7B 05      46 BIORET LDA OURCH      ;get newest cursor position
C8C8:2C 1F C0      47      BIT RD80VID    ;IN 80-MODE?
C8CB:10 02 C8CF    48      BPL SETALL    ;=>no, set other cursors
C8CD:A9 00         49      LDA #0      ;pin CH to 0 for 80 columns
C8CF:85 24         50 SETALL STA CH
C8D1:8D 7B 04      51      STA OLDCH    ;REMEMBER THE SETTING
C8D4:68           52 GETREGS PLA      ;RESTORE
C8D5:AA           53      TAX
C8D6:68           54      PLA          ;X AND Y
C8D7:A8           55      TAY
C8D8:AD 7B 06      56      LDA CHAR
C8DB:60           57      RTS          ;RETURN TO BASIC
C8DC:              25      INCLUDE BINPUT
C8DC:              1 *
C8DC:              2 * BASIC input entry point called by entry point in the
C8DC:              3 * $C3 space. This is the way things normally happen.
C8DC:              4 *
C8DC:A4 24        5 BINPUT LDY CH

```

```

C8DE:AD 7B 06      6      LDA  CHAR
C8E1:91 28        7      STA  (BASL),Y
C8E3:20 50 C8      8      JSR  CSETUP    ;get newest cursor
C8E6:20 26 CE      9 B.INPUT JSR  INVERT    ;invert that char
C8E9:20 3B C8     10     JSR  GETKEY    ;GET A KEY
C8EC:8D 7B 06     11     STA  CHAR    ;SAVE IT
C8EF:20 26 CE     12     JSR  INVERT    ;REMOVE CURSOR
C8F2:A8           13     TAY      ;preserve acc.
C8F3:           14 *
C8F3:           15 * On pure input, an uninterpreted character code should
C8F3:           16 * be returned. If M.CTL is set, however, escape functions
C8F3:           17 * are enabled, and CTL-U causes the character under the
C8F3:           18 * cursor to be picked up from the screen.
C8F3:           19 * M.CTL is set whenever a character is requested using
C8F3:           20 * RDCHAR in the $F8 ROM.
C8F3:           21 *
C8F3:AD FB 04     22     LDA  MODE    ;is escape mode enabled?
C8F6:29 08        23     AND  #M.CTL
C8F8:F0 CB C8C5   24     BEQ  BIORET    ;=>no,return
C8FA:C0 8D        25     CPY  #$8D    ;was it a CR
C8FC:D0 08 C906   26     BNE  NOTACR    ;=>nope, not a CR
C8FE:AD FB 04     27     LDA  MODE
C901:29 F7        28     AND  #255-M.CTL ;else end of line...
C903:8D FB 04     29     STA  MODE    ; disable escape
C906:      C906   30 NOTACR EQU  *
C906:C0 9B        31     CPY  #$9B    ;ESCAPE KEY?
C908:F0 11 C91B   32     BEQ  ESCAPING ;=>YES IT IS
C90A:           33 *
C90A:           34 * Not an escape sequence. Check for control-u.
C90A:           35 *
C90A:C0 95        36     CPY  #$95    ;is it control-U?
C90C:D0 B7 C8C5   37     BNE  BIORET    ;no, return to caller
C90E:AC 7B 05     38     LDY  OURCH    ;get horizontal position
C911:20 44 CE     39     JSR  PICK    ;and pick up the char
C914:09 80        40     ORA  #$80    ;always pick as normal
C916:8D 7B 06     41     STA  CHAR    ;save keystroke
C919:D0 AA C8C5   42     BNE  BIORET    ;=>(always) return to caller
C91B:           43 *
C91B:           44 * Start an escape sequence. If the next character
C91B:           45 * pressed is one of the following, it is executed.
C91B:           46 * Otherwise it is ignored.
C91B:           47 *
C91B:           48 * @ - home & clear
C91B:           49 * E - clear to end of line
C91B:           50 * F - clear to end of screen
C91B:           51 * I - move cursor up
C91B:           52 * J - move cursor left
C91B:           53 * K - move cursor right
C91B:           54 * M - move cursor down
C91B:           57 * 4 - enter 40 column mode
C91B:           58 * 8 - enter 80 column mode
C91B:           59 * CTL-D- disable the printing of control characters
C91B:           60 * CTL-E- enable the printing of control characters
C91B:           61 * CTL-Q- quit (PR#0/IN#0)

```



```

C91B:      62 *   The four arrow keys (as IJKM)
C91B:      63 *
C91B:      64      MSB OFF
C91B:      C91B 65 ESCAPING EQU *
C91B:20 B1 CE 66      JSR ESCON      ;ESCAPE CURSOR ON
C91E:20 3B C8 67      JSR GETKEY    ;GET ESCAPE FUNCTION
C921:20 C4 CE 68      JSR ESCOFF    ;REPLACE ORIGINAL CHARACTER
C924:20 14 CE 69      JSR UPSHFT    ;upshift the char
C927:29 7F    70      AND #$7F      ;DROP HI BIT
C929:A0 10    71      LDY #ESCNUM-1 ;COUNT/INDEX
C92B:D9 7C C9 72 ESC2  CMP ESCTAB,Y  ;IS IT A VALID ESCAPE?
C92E:F0 05 C935 73      BEQ ESC3     ;=>YES
C930:88      74      DEY
C931:10 F8 C92B 75      BPL ESC2     ;TRY 'EM ALL...
C933:30 0F C944 76      BMI ESCSPEC  ;=>MAYBE IT'S A SPECIAL ONE
C935:      77 *
C935:      C935 78 ESC3  EQU *
C935:B9 6B C9 79      LDA ESCCHAR,Y ;GET CHAR TO "PRINT"
C938:29 7F    80      AND #$7F      ;DROP HI BIT (FLAG)
C93A:20 D6 CA 81      JSR CTLCHAR    ;EXECUTE IT
C93D:B9 6B C9 82      LDA ESCCHAR,Y ;GET FLAG
C940:30 D9 C91B 83      BMI ESCAPING ;=>STAY IN ESCAPE MODE
C942:10 A2 C8E6 84      BPL B.INPUT  ;=>QUIT ESCAPE MODE
C944:      85 *
C944:      C944 86 ESCSPEC EQU *
C944:A8      87      TAY            ;put char here
C945:AD FB 04 88      LDA MODE      ;so we can put this here
C948:C0 11    89      CPY #$11      ;was it Quit?
C94A:D0 0B C957 90      BNE ESCSP1    ;=>no
C94C:20 4D CD 91      JSR X.NAK      ;do the quitting stuff
C94F:A9 98    92      LDA #$98      ;make it look like
C951:8D 7B 06 93      STA CHAR      ;CTL-X was pressed
C954:4C C5 C8 94      JMP BIORET    ;=>quit the card forever
C957:      95 *
C957:C0 05    96 ESCSP1 CPY #$05      ;was it CTL-E for enable
C959:D0 08 C963 97      BNE ESCSP4    ;=>no
C95B:29 DF    98      AND #255-M.CTL2 ;yes, enable ctl chars
C95D:8D FB 04 99 ESCSP2 STA MODE      ;save new mode
C960:4C E6 C8 100 ESCSP3 JMP B.INPUT  ;=> exit escape mode
C963:      101 *
C963:C0 04    102 ESCSP4 CPY #$04      ;was it CTL-D for disable
C965:D0 F9 C96D 103      BNE ESCSP3    ;=>no, exit escape mode
C967:09 20    104      ORA #M.CTL2     ;disable ctl chars
C969:D0 F2 C95D 105      BNE ESCSP2    ;=> exit escape mode
C96B:      106 *
C96B:      107 * This table contains the control characters which,
C96B:      108 * when executed, carry out the escape functions. If
C96B:      109 * the high bit of the character is set, it means that
C96B:      110 * escape mode should not be exited after execution of
C96B:      111 * the character.
C96B:      112 *
C96B:      C96B 113 ESCCHAR EQU *
C96B:OC      114      DFB $0C          ;@: FORMFEED
C96C:1C      115      DFB $1C          ;A: FS

```



```

C96D:08      116      DFB $08      ;B: BS
C96E:0A      117      DFB $0A      ;C: LF
C96F:1F      118      DFB $1F      ;D: US
C970:1D      119      DFB $1D      ;E: GS
C971:0B      120      DFB $0B      ;F: VT
C972:9F      121      DFB $1F+$80   ;I: US (STAY ESC)
C973:88      122      DFB $08+$80   ;J: BS (STAY ESC)
C974:9C      123      DFB $1C+$80   ;K: FS (STAY ESC)
C975:8A      124      DFB $0A+$80   ;M: LF (STAY ESC)
C976:11      125      DFB $11      ;4 :DC1
C977:12      126      DFB $12      ;8 :DC2
C978:88      127      DFB $08+$80   ;<-:BS (STAY ESC)
C979:8A      128      DFB $0A+$80   ;DN:LF (STAY ESC)
C97A:9F      129      DFB $1F+$80   ;UP:US (STAY ESC)
C97B:9C      130      DFB $1C+$80   ;->:FS (STAY ESC)
C97C:        131      *
C97C:        132      MSB OFF      ;high bit already masked
C97C:        C97C 133 ESCTAB EQU *
C97C:40      134      ASC '@'
C97D:41      135      ASC 'A'      ;HANDLE OLD ESCAPES
C97E:42      136      ASC 'B'
C97F:43      137      ASC 'C'
C980:44      138      ASC 'D'
C981:45      139      ASC 'E'
C982:46      140      ASC 'F'
C983:49      141      ASC 'I'
C984:4A      142      ASC 'J'
C985:4B      143      ASC 'K'
C986:4D      144      ASC 'M'
C987:34      145      ASC '4'
C988:38      146      ASC '8'
C989:08      147      DFB $08      ;LEFT ARROW
C98A:0A      148      DFB $0A      ;DOWN ARROW
C98B:0B      149      DFB $0B      ;UP ARROW
C98C:15      150      DFB $15      ;RITE ARROW
C98D:        0011 151 ESCNUM EQU *-ESCTAB
C98D:        152      MSB ON
C98D:        153      *
C98D:        154      * Tack on diag 128K test here
C98D:        155      *
C98D:2C 13 C0 156 STAUX BIT RDRAMRD ;aux done yet?
C990:30 11 C9A3 157 BMI XSTAUX ;=>yes, exit
C992:A9 EE      158 LDA #$EE ;get test pattern
C994:8D 05 C0   159 STA WRCARDRAM ;write AUX RAM
C997:8D 03 C0   160 STA RDCARDRAM ;read AUX RAM
C99A:8D 00 0C   161 STA $C00 ;test this byte
C99D:8D 00 08   162 STA $800 ;and this is 1K off
C9A0:CD 00 0C   163 CMP $C00 ;has $C00 been updated?
C9A3:60         164 XSTAUX RTS ;check in main diags.
C9A4:         165      *
C9A4:         166      * ESCOUT used by ESCFIX in $C1 page
C9A4:         167      *
C9A4:         168      MSB ON
C9A4:CA CB CD C9 169 ESCOUT ASC 'JKMI' ;The arrows

```

```

C9A8:      170      MSB OFF
C9A8:      26      INCLUDE PASCAL
C9A8:      1 *****
C9A8:      2 * PASCAL 1.0 OUTPUT HOOK:
C9A8:      3 *****
C9A8:      4      DS      C8ORG+$1AA-*,0
C9AA:      0002    5      IFNE *-C8ORG-$1AA
S      0000    6      FAIL 2,'C9AA  HOOK ALIGNMENT'
C9AA:      7      FIN
C9AA:AD 7B 06    8      LDA  CHAR      ;GET OUTPUT CHARACTER
C9AD:4C 56 C3    9      JMP  JPWRITE  ;=>USE STANDARD WRITE
C9B0:      10 *****
C9B0:      11 *
C9B0:      12 *****
C9B0:      13 * PASCAL INITIALIZATION:
C9B0:      14 * Disable printing of mouse text
C9B0:      15 *****
C9B0:      C9B0  16 PINIT1.0 EQU *
C9B0:A9 83      17      LDA  #M.PASCAL+M.PAS1.O+M.MOUSE
C9B2:D0 02    C9B6  18      BNE  PINIT2      ;=>always
C9B4:      C9B4  19 PINIT  EQU  *
C9B4:A9 81      20      LDA  #M.PASCAL+M.MOUSE ;SAY WE'RE
C9B6:      21 *
C9B6:      C9B6  22 PINIT2 EQU  *
C9B6:48      23      PHA      ;save version ID
C9B7:      24 *
C9B7:      25 * SEE IF THE CARD'S PLUGGED IN:
C9B7:      26 *
C9B7:20 90 CA      27      JSR  TESTCARD ;IS IT THERE?
C9BA:F0 04    C9C0  28      BEQ  PIGOOD  ;=>YES
C9BC:68      29      PLA      ;discard ID byte
C9BD:A2 09      30      LDX  #9      ;IORESULT='NO DEVICE'
C9BF:60      31      RTS
C9C0:      32 *
C9C0:      C9C0  33 PIGOOD EQU  *
C9C0:68      34      PLA      ;get version ID
C9C1:8D FB 04      35      STA  MODE      ; and save it
C9C4:8D 01 C0      36      STA  SET80COL ;ENABLE 80 STORE
C9C7:8D 0D C0      37      STA  SET80VID ; AND 80 VIDEO
C9CA:8D 0F C0      38      STA  SETALTCHAR ;NORM+INV LCASE
C9CD:20 D4 CE      39      JSR  PSETUP ;set window and cursor
C9D0:20 90 CC      40      JSR  X.FF ;HOME & CLEAR IT
C9D3:4C 1F CA      41      JMP  DOBASL ;fix OLDBASL/H, display cursor, exit
C9D6:      42 *****
C9D6:      43 * PASCAL INPUT:
C9D6:      44 *
C9D6:      45 * Character always returned with high bit clear.
C9D6:      46 *
C9D6:      47 *****
C9D6:      C9D6  48 PREAD  EQU  *
C9D6:20 D4 CE      49      JSR  PSETUP ;SETUP ZP STUFF
C9D9:20 3B C8      50      JSR  GETKEY ;GET A KEYSTROKE
C9DC:29 7F      51      AND  #$7F ;DROP HI BIT
C9DE:8D 7B 06      52      STA  CHAR ;SAVE THE CHAR

```

```

C9E1:A2 00      53      LDX #0      ;IORESULT='GOOD'
C9E3:AD FB 04   54      LDA MODE    ;ARE WE IN 1.0-MODE?
C9E6:29 02      55      AND #M.PAS1.0
C9E8:F0 02      C9EC    56      BEQ PREADRET2 ;=>NOPE
C9EA:A2 C3      57      LDX #<CN00    ;YES, RETURN CN IN X
C9EC:           58 *
C9EC:           C9EC    59 PREADRET2 EQU *
C9EC:AD 7B 06   60      LDA CHAR    ;RESTORE CHAR
C9EF:60         61      RTS
C9F0:           62 *
C9F0:           63 * PASCAL OUTPUT:
C9F0:           64 * Note: to be executed, control characters must have
C9F0:           65 * their high bits cleared. All other characters are
C9F0:           66 * displayed regardless of their high bits.
C9F0:           67 *
C9F0:           C9F0    68 PWRITE EQU *
C9F0:29 7F      69      AND #$7F      ;clear high bits
C9F2:AA         70      TAX      ;save character
C9F3:20 D4 CE    71      JSR PSETUP ;SETUP ZP STUFF, don't set ROM
C9F6:A9 08      72      LDA #M.GOXY ;ARE WE DOING GOTOXY?
C9F8:2C FB 04   73      BIT MODE
C9FB:D0 32      CA2F    74      BNE GETX      ;=>Doing X or Y?
C9FD:8A         75      TXA      ;now check for control char
C9FE:2C 2E CA    76      BIT PRTS    ;is it control?
CA01:F0 50      CA53    77      BEQ PCTL     ;=>yes, do control
CA03:AC 7B 05   78      LDY OURCH    ;get horizontal position
CA06:24 32      79      BIT INVFLG   ;check for inverse
CA08:10 02      CA0C    80      BPL PWRI    ;inverse, go store it
CA0A:09 80      81      ORA #$80
CA0C:20 70 CE    82 PWRI    JSR STORIT ;now store it (erasing cursor)
CA0F:C8         83      INY      ;INC CH
CA10:8C 7B 05   84      STY OURCH
CA13:C4 21      85      CPY WNDWDTH
CA15:90 08      CA1F    86      BCC DOBASL
CA17:A9 00      87      LDA #0      ;do carriage return
CA19:8D 7B 05   88      STA OURCH
CA1C:20 D8 CB    89      JSR X.LF     ;and linefeed
CA1F:A5 28      90 DOBASL LDA BASL   ;save BASL for pascal
CA21:8D 7B 07   91      STA OLDBASL
CA24:A5 29      92      LDA BASH
CA26:8D FB 07   93      STA OLDBASH
CA29:20 1F CE    94 PWRITERET JSR PASINV ;display new cursor
CA2C:A2 00      95 PRET   LDX #$0     ;return with no error
CA2E:60         96      PRTS    RTS
CA2F:           97 *
CA2F:           98 * HANDLE GOTOXY STUFF:
CA2F:           99 *
CA2F:20 1F CE    100 GETX   JSR PASINV ;turn off cursor
CA32:8A         101      TXA      ;get character
CA33:38         102      SEC
CA34:E9 20      103      SBC #32    ;MAKE BINARY
CA36:2C FB 06   104      BIT XCOORD ;doing X?
CA39:30 30      CA6B    105      BMI PSETX   ;=>yes, set it
CA3B:           106 *

```

```

CA3B:          107 * Set Y and do the GOTOXY
CA3B:          108 *
CA3B:8D FB 05  109 GETY   STA   OURCV
CA3E:85 25     110        STA   CV
CA40:20 BA CA  111        JSR   BASCALC    ;calc base addr
CA43:AD FB 06  112        LDA   XCOORD
CA46:8D 7B 05  113        STA   OURCH      ;set cursor horizontal
CA49:A9 F7     114        LDA   #255-M.GOXY ;turn off gotoxy
CA4B:2D FB 04  115        AND   MODE
CA4E:8D FB 04  116        STA   MODE
CA51:D0 CC CA1F 117        BNE   DOBASL    ;=>DONE (ALWAYS TAKEN)
CA53:          118 *
CA53:20 1F CE  119 PCTL   JSR   PASINV     ;turn off cursor
CA56:8A        120        TXA           ;get char
CA57:C9 1E     121        CMP   #$1E      ;is it gotoXY?
CA59:F0 06 CA61 122        BEQ   STARTXY    ;=>yes, start it up
CA5B:20 D6 CA  123        JSR   CTLCHAR    ;EXECUTE IT IF POSSIBLE
CA5E:4C 1F CA  124        JMP   DOBASL    ;=>update BASL/H, cursor, exit
CA61:          125 *
CA61:          126 * START THE GOTOXY SEQUENCE:
CA61:          127 *
CA61:          CA6 1 128 STARTXY EQU   *
CA61:A9 08     129        LDA   #M.GOXY
CA63:0D FB 04  130        ORA   MODE      ;turn on gotoxy
CA66:8D FB 04  131        STA   MODE
CA69:A9 FF     132        LDA   #$FF      ;set XCOORD to -1
CA6B:8D FB 06  133 PSETX  STA   XCOORD    ;set X
CA6E:4C 29 CA  134        JMP   PWRITERET ;=>display cursor and exit
CA71:          27        INCLUDE SUBS1
CA71:          CA7 1 1  DOMN   EQU   *
CA71:AA        2        TAX
CA72:A5 2A     3        LDA   BAS2L      ;SAVE IT
CA74:A0 03     4        LDY   #$03      ;GET OPCODE AGAIN
CA76:E0 8A     5        CPX   #$8A
CA78:F0 0B CA8 5 6        BEQ   MNNDX3
CA7A:4A        7 MNNDX1 LSR   A
CA7B:90 08 CA8 5 8        BCC   MNNDX3    ;FORM INDEX INTO MNEMONIC TABLE
CA7D:4A        9        LSR   A
CA7E:4A       10 MNNDX2 LSR   A          ; 1) 1XXX1010 => 00101XXX
CA7F:09 20     11        ORA   #$20      ; 2) XXXYYY01 => 00111XXX
CA81:88       12        DEY           ; 3) XXXYYY10 => 00110XXX
CA82:D0 FA CA7 E 13        BNE   MNNDX2    ; 4) XXXYY100 => 00100XXX
CA84:C8       14        INY           ; 5) XXXXX000 => 000XXXXX
CA85:88       15 MNNDX3 DEY
CA86:D0 F2 CA7 A 16        BNE   MNNDX1
CA88:60       17        RTS
CA89:          18 *
CA89:          19 * Switch in slot 3, then test for a ROM card.
CA89:          20 * If none found, test for 80 column card,
CA89:          21 * else return with BNE.
CA89:          22 *
CA89:          CA8 9 23 TSTROMCRD EQU *
CA89:20 B7 F8  24        JSR   TSTROM     ;test for ROM card
CA8C:D0 02 CA9 0 25        BNE   TESTCARD ;=>no ROM, check for 80 column card

```

```

CA8E:C8      26      INY      ;make BNE for return
CA8F:60      27      RTS
CA90:        28      *
CA90:        29      *****
CA90:        30      * NAME      : TESTCARD
CA90:        31      * FUNCTION: SEE IF 80COL CARD PLUGGED IN
CA90:        32      * INPUT   : NONE
CA90:        33      * OUTPUT  : 'BEQ' IF CARD AVAILABLE
CA90:        34      *          : 'BNE' IF NOT
CA90:        35      * VOLATILE: AC,Y
CA90:        36      *****
CA90:        37      *
CA90:        CA90    38      TESTCARD EQU *
CA90:AD 1C CO    39      LDA      RDPAGE2      ;REMEMBER CURRENT VIDEO DISPLAY
CA93:0A        40      ASL      A              ; IN THE CARRY
CA94:A9 88      41      LDA      #$88          ;USEFUL CHAR FOR TESTING
CA96:2C 18 CO    42      BIT      RD80COL      ;REMEMBER VIDEO MODE IN 'N'
CA99:8D 01 CO    43      STA      SET80COL      ;ENABLE 80COL STORE
CA9C:08        44      PHP                      ;SAVE 'N' AND 'C' FLAGS
CA9D:8D 55 CO    45      STA      TXTPAGE2      ;SET PAGE2
CAA0:AC 00 04    46      LDY      $0400          ;GET FIRST CHAR
CAA3:8D 00 04    47      STA      $0400          ;SET TO A '*'
CAA6:AD 00 04    48      LDA      $0400          ;GET IT BACK FROM RAM
CAA9:8C 00 04    49      STY      $0400          ;RESTORE ORIG CHAR
CAAC:28        50      PLP                      ;RESTORE 'N' AND 'C' FLAGS
CAAD:B0 03 CAB2  51      BCS      STAY2          ;STAY IN PAGE2
CAAF:8D 54 CO    52      STA      TXTPAGE1      ;RESTORE PAGE1
CAB2:        CAB2  53      STAY2      EQU      *
CAB2:30 03 CAB7  54      BMI      STAY80          ;=>STAY IN 80COL MODE
CAB4:8D 00 CO    55      STA      CLR80COL      ;TURN OFF 80COL STORE
CAB7:        CAB7  56      STAY80      EQU      *
CAB7:C9 88      57      CMP      #$88          ;WAS CHAR VALID?
CAB9:60        58      RTS                      ;RETURN RESULT AS BEQ/BNE
CABA:        59      *
CABA:        60      * Do the
normal monitor ROM BASCALC
CABA:        61      *
CABA:        CABA  62      BASCALC EQU *
CABA:48        63      PHA
CABB:4A        64      LSR      A
CABC:29 03      65      AND      #$03
CABE:09 04      66      ORA      #$04
CAC0:85 29      67      STA      BASH
CAC2:68        68      PLA
CAC3:29 18      69      AND      #$18
CAC5:90 02 CAC9  70      BCC      BSCLC2
CAC7:69 7F      71      ADC      #$7F
CAC9:85 28      72      BSCLC2 STA      BASL
CACB:0A        73      ASL      A
CACC:0A        74      ASL      A
CACD:05 28      75      ORA      BASL
CACF:85 28      76      STA      BASL
CAD1:60        77      RTS
CAD2:        78      *

```

```

CAD2:      79 *****
CAD2:      80 * NAME      : CTLCHARO
CAD2:      81 * FUNCTION: Execute CTL char if M.CTL=0
CAD2:      82 * INPUT   : AC=CHAR
CAD2:      83 * OUTPUT  : 'BCS' if not executed
CAD2:      84 *        : 'BCC' if executed
CAD2:      85 * VOLATILE: NOTHING
CAD2:      86 * CALLS   : MANY THINGS
CAD2:      87 *****
CAD2:      88 *
CAD2:2C 06 CB 89 CTLCHARO BIT SEV1      ;set V (use M.CTL)
CAD5:50 FE CAD5 90 BVC *              ;skip CLC
CAD6:      CAD6 91 ORG *-1
CAD6:      92 *
CAD6:      93 *****
CAD6:      94 * NAME      : CTLCHAR
CAD6:      95 * FUNCTION: Always execute CTL char
CAD6:      96 * INPUT   : AC=CHAR
CAD6:      97 * OUTPUT  : 'BCS' if not executed
CAD6:      98 *        : 'BCC' if ctl executed
CAD6:      99 * VOLATILE: NOTHING
CAD6:     100 * CALLS   : MANY THINGS
CAD6:     101 *****
CAD6:     102 *
CAD6:B8      103 CTLCHAR CLV          ;clear V (ignore M.CTL)
CAD7:8D 7B 07 104 STA TEMP1          ;TEMP SAVE OF CHAR
CADA:48      105 PHA                  ;SAVE AC
CADB:98      106 TYA                  ;SAVE Y
CADC:48      107 PHA
CADD:      108 *
CADD:AC 7B 07 109 LDY TEMP1          ;GET CHAR IN QUESTION
CAE0:C0 05    110 CPY #$05          ;IS IT NUL..EOT?
CAE2:90 13 CAF7 111 BCC CTLCHARX    ;=>YES, NOT USED
CAE4:B9 B4 CB 112 LDA CTLADH-5,Y ;Get high byte of address
CAE7:F0 0E CAF7 113 BEQ CTLCHARX    ;=>ctl not implemented
CAE9:50 12 CAFD 114 BVC CTLG00      ;=> CLTCHAR: always execute
CAEB:      115 *
CAEB:      0000 116 DO TEST
S      117 BPL CTLG00      ;=>CR,BEL,LF,BS always done
CAEB:      118 ELSE
CAEB:30 10 CAFD 119 BMI CTLG00      ;=>CR,BEL,LF,BS always done
CAED:      120 FIN
CAED:      121 *
CAED:8D 7B 07 122 STA TEMP1          ;save high byte of address
CAFO:AD FB 04 123 LDA MODE          ;if control chars
CAF3:29 28    124 AND #M.CTL+M.CTL2 ;are enabled
CAF5:F0 03 CAF7 125 BEQ CTLG00      ;=>then go do them
CAF7:      126 *
CAF7:      CAF7 127 CTLCHARX EQU *
CAF7:38      128 SEC              ;SAY 'NOT CTL'
CAF8:B0 09 CB03 129 BCS CTLRET      ;=>DONE
CAFA:      130 *
CAFA:AD 7B 07 131 CTG0 LDA TEMP1          ;get address back
CAFD:      CAFD 132 CTG00 EQU *

```

```

CAFD:      0000 133      DO    TEST
S          134      AND    #$7F      ;for test, hi bit clear
CAFD:      135      ELSE
CAFD:09 80  136      ORA    #$80      ;hi bit always set
CAFF:      137      FIN
CAFF:20 07 CB 138      JSR    CTLXFER  ;EXECUTE SUBROUTINE
CB02:      139 *
CB02:18     140      CLC          ;SAY 'CTL CHAR EXECUTED'
CB03:      CB03 141 CTLRET EQU *
CB03:68     142      PLA          ;RESTORE
CB04:A8     143      TAY          ; Y
CB05:68     144      PLA          ; AND AC
CB06:60     145 SEV1  RTS
CB07:      146 *
CB07:      CB07 147 CTLXFER EQU *
CB07:48     148      PHA          ;PUSH ONTO STACK FOR
CB08:B9 99 CB 149      LDA    CTLADL-5,Y ; TRANSFER TRICK
CB0B:48     150      PHA
CB0C:60     151      RTS          ;XFER TO ROUTINE
CB0D:      152 *
CB0D:      153 * Turn cursor on for Pascal only
CB0D:      154 *
CB0D:AD FB 04 155 X.CUR.ON LDA MODE      ;get mode byte
CB10:10 05 CB17 156      BPL    CURON.X ;=>not pascal, don't do it
CB12:29 EF  157      AND    #255-M.CURSOR ;clear cursor bit
CB14:8D FB 04 158 SAVCUR STA MODE      ;save it
CB17:60     159 CURON.X RTS          ;and exit
CB18:      160 *
CB18:      161 * Turn cursor off for Pascal only.
CB18:      162 * Cursor is not displayed during call.
CB18:      163 *
CB18:AD FB 04 164 X.CUR.OFF LDA MODE      ;get mode byte
CB1B:10 FA CB17 165      BPL    CURON.X ;=>not pascal, don't do it
CB1D:09 10  166      ORA    #M.CURSOR ;turn on cursor bit
CB1F:D0 F3 CB14 167      BNE    SAVCUR ;save and exit
CB21:      168 *
CB21:      169 * EXECUTE BELL:
CB21:      170 *
CB21:      CB21 171 X.BELL EQU *
CB21:A9 40  172      LDA    #$40      ;RIPPED OFF FROM MONITOR
CB23:20 34 CB 173      JSR    WAIT
CB26:A0 C0  174      LDY    #$C0
CB28:A9 0C  175 BELL2  LDA    #$0C
CB2A:20 34 CB 176      JSR    WAIT
CB2D:AD 30 C0 177      LDA    SPKR
CB30:88     178      DEY
CB31:D0 F5 CB28 179      BNE    BELL2
CB33:60     180      RTS
CB34:      181 *
CB34:      CB34 182 WAIT EQU *      ;RIPPED OFF FROM MONITOR ROM
CB34:38     183      SEC
CB35:48     184 WAIT2 PHA
CB36:E9 01  185 WAIT3 SBC    #1
CB38:D0 FC CB36 186      BNE    WAIT3

```

CB3A:68		187	PLA	
CB3B:E9 01		188	SBC #1	
CB3D:D0 F6 CB35		189	BNE WAIT2	
CB3F:60		190	RTS	
CB40:		191 *		
CB40:		192 *	EXECUTE BACKSPACE:	
CB40:		193 *		
CB40:	CB40	194 X.BS	EQU *	
CB40:CE 7B 05		195	DEC OURCH	;BACK UP CH
CB43:10 0B CB50		196	BPL BSDONE	;=>DONE
CB45:A5 21		197	LDA WNDWDTH	;BACK UP TO PRIOR LINE
CB47:8D 7B 05		198	STA OURCH	;SET CH
CB4A:CE 7B 05		199	DEC OURCH	
CB4D:20 79 CB		200	JSR X.US	;NOW DO REV LINEFEED
CB50:	CB50	201 BSDONE	EQU *	
CB50:60		202	RTS	
CB51:		203 *		
CB51:		204 *	EXECUTE CARRIAGE RETURN:	
CB51:		205 *		
CB51:	CB51	206 X.CR	EQU *	
CB51:A9 00		207	LDA #0	;BACK UP CH TO
CB53:8D 7B 05		208	STA OURCH	; BEGINNING OF LINE
CB56:AD FB 04		209	LDA MODE	;ARE WE IN BASIC?
CB59:30 03 CB5E		210	BMI X.CRRET	;=> Pascal, avoid auto LF
CB5B:20 D8 CB		211	JSR X.LF	;EXECUTE AUTO LF FOR BASIC
CB5E:	CB5E	212 X.CRRET	EQU *	
CB5E:60		213	RTS	
CB5F:		214 *		
CB5F:		215 *	EXECUTE HOME:	
CB5F:		216 *		
CB5F:	CB5F	217 X.EM	EQU *	
CB5F:A5 22		218	LDA WNDTOP	
CB61:85 25		219	STA CV	
CB63:A9 00		220	LDA #0	
CB65:8D 7B 05		221	STA OURCH	;STUFF CH
CB68:4C FE CD		222	JMP VTAB	;set base for OURCV
CB6B:		223 *		
CB6B:		224 *	EXECUTE FORWARD SPACE:	
CB6B:		225 *		
CB6B:	CB6B	226 X.FS	EQU *	
CB6B:EE 7B 05		227	INC OURCH	;BUMP CH
CB6E:AD 7B 05		228	LDA OURCH	;GET THE POSITION
CB71:C5 21		229	CMP WNDWDTH	;OFF THE RIGHT SIDE?
CB73:90 03 CB78		230	BCC X.FSRET	;=>NO, GOOD
CB75:20 51 CB		231	JSR X.CR	;=>YES, WRAP AROUND
CB78:		232 *		
CB78:	CB78	233 X.FSRET	EQU *	
CB78:60		234	RTS	
CB79:		235 *		
CB79:		236 *	EXECUTE REVERSE LINEFEED:	
CB79:		237 *		
CB79:A5 22		238 X.US	LDA WNDTOP	;are we at top?
CB7B:C5 25		239	CMP CV	
CB7D:B0 1E CB9D		240	BCS X.USRET	;=>yes, stay there


```

CB7F:C6 25      241      DEC CV      ;else go up a line
CB81:4C FE CD   242      JMP VTAB    ;exit thru VTAB (update OURCV)
CB84:           243 *
CB84:           244 * EXECUTE "NORMAL VIDEO"
CB84:           245 *
CB84:           CB84 246 X.SO      EQU *
CB84:AD FB 04   247      LDA MODE      ;SET MODE BIT
CB87:10 02      CB8B 248      BPL X.S01    ;don't set mode for BASIC
CB89:29 FB      249      AND #255-M.VMODE ;SET 'NORMAL'
CB8B:A0 FF      250 X.S01      LDY #255
CB8D:D0 09      CB98 251      BNE STUFFINV ;(ALWAYS)
CB8F:           252 *
CB8F:           253 * EXECUTE "INVERSE VIDEO"
CB8F:           254 *
CB8F:           CB8F 255 X.SI      EQU *
CB8F:AD FB 04   256      LDA MODE      ;SET MODE BIT
CB92:10 02      CB96 257      BPL X.SI1    ;don't set mode for BASIC
CB94:09 04      258      ORA #M.VMODE    ;SET 'INVERSE'
CB96:A0 7F      259 X.SI1      LDY #127
CB98:8D FB 04   260 STUFFINV STA MODE      ;SET MODE
CB9B:84 32      261      STY INVFLG    ;STUFF FLAG TOO
CB9D:60         262 X.USRET      RTS
CB9E:           263 *
CB9E:           CB9E 264 CTLADL      EQU *
CB9E:0C         265      DFB #>X.CUR.ON-1 ;ENQ
CB9F:17         266      DFB #>X.CUR.OFF-1 ;ACK
CBA0:20         267      DFB #>X.BELL-1 ;BEL
CBA1:3F         268      DFB #>X.BS-1 ;BS
CBA2:00         269      DFB 0 ;HT
CBA3:D7         270      DFB #>X.LF-1 ;LF
CBA4:73         271      DFB #>X.VT-1 ;VT
CBA5:8F         272      DFB #>X.FF-1 ;FF
CBA6:50         273      DFB #>X.CR-1 ;CR
CBA7:83         274      DFB #>X.SO-1 ;SO
CBA8:8E         275      DFB #>X.SI-1 ;SI
CBA9:00         276      DFB 0 ;DLE
CBAA:E9         277      DFB #>X.DC1-1 ;DC1
CBAB:FB         278      DFB #>X.DC2-1 ;DC2
CBAC:00         279      DFB 0 ;DC3
CBAD:00         280      DFB 0 ;DC4
CBAE:4C         281      DFB #>X.NAK-1 ;NAK
CBAF:D3         282      DFB #>SCROLLDN-1 ;SYN
CBB0:EA         283      DFB #>SCROLLUP-1 ;ETB
CBB1:3C         284      DFB #>MOUSEOFF-1
CBB2:5E         285      DFB #>X.EM-1 ;EM
CBB3:95         286      DFB #>X.SUB-1 ;SUB
CBB4:43         287      DFB #>MOUSEON-1
CBB5:6A         288      DFB #>X.FS-1 ;FS
CBB6:99         289      DFB #>X.GS-1 ;GS
CBB7:00         290      DFB 0 ;RS
CBB8:78         291      DFB #>X.US-1 ;US
CBB9:           292 *
CBB9:           CBB9 293 CTLADH      EQU *
CBB9:4B         294      DFB #<X.CUR.ON-$8001 ;ENQ

```

CBBA:4B	295	DFB	#<X.CUR.OFF-\$8001 ;ACK
CBBB:CB	296	DFB	#<X.BELL-1 ;BEL
CBBC:CB	297	DFB	#<X.BS-1 ;BS
CBBD:00	298	DFB	0 ;HT
CBBE:CB	299	DFB	#<X.LF-1 ;LF
CBBF:4C	300	DFB	#<X.VT-\$8001 ;VT
CB0:4C	301	DFB	#<X.FF-\$8001 ;FF
CBC1:CB	302	DFB	#<X.CR-1 ;CR
CBC2:4B	303	DFB	#<X.S0-\$8001 ;S0
CBC3:4B	304	DFB	#<X.SI-\$8001 ;SI
CBC4:00	305	DFB	0 ;DLE
CBC5:4C	306	DFB	#<X.DC1-\$8001 ;DC1
CBC6:4C	307	DFB	#<X.DC2-\$8001 ;DC2
CBC7:00	308	DFB	0 ;DC3
CBC8:00	309	DFB	0 ;DC4
CBC9:4D	310	DFB	#<X.NAK-\$8001 ;NAK
CBCA:4B	311	DFB	#<SCROLLDN-\$8001 ;SYN
CBCB:4B	312	DFB	#<SCROLLUP-\$8001 ;ETB
CBCC:4D	313	DFB	#<MOUSEOFF-\$8001
CBCD:4B	314	DFB	#<X.EM-\$8001 ;EM
CBCE:4C	315	DFB	#<X.SUB-\$8001 ;SUB
CBCF:4D	316	DFB	#<MOUSEON-\$8001
CBD0:4B	317	DFB	#<X.FS-\$8001 ;FS
CBD1:4C	318	DFB	#<X.GS-\$8001 ;GS
CBD2:00	319	DFB	0 ;RS
CBD3:4B	320	DFB	#<X.US-\$8001 ;US
CBD4:	28	INCLUDE	SUBS2
CBD4:	1	*	
CBD4:	2	*	SCROLLIT scrolls the screen either up or down, depending
CBD4:	3	*	on the value of X. It scrolls within windows with even
CBD4:	4	*	or odd edges for both 40 and 80 columns. It can scroll
CBD4:	5	*	windows down to 1 characters wide.
CBD4:	6	*	
CBD4:A0 00	7	SCROLLDN LDY #0	;direction = down
CBD6:F0 15 CBED	8	BEQ SCROLLIT	;=>go do scroll
CBD8:	9	*	
CBD8:	10	*	EXECUTE LINEFEED:
CBD8:	11	*	
CBD8: CBDS	12	X.LF EQU *	
CBD8:E6 25	13	INC CV	
CBDA:A5 25	14	LDA CV	;SEE IF OFF BOTTOM
CBDC:8D FB 05	15	STA OURCV	
CBDF:C5 23	16	CMP WNDBTM	;OFF THE END?
CBE1:B0 03 CBE6	17	BCS X.LF2	;=>yes, scroll screen
CBE3:4C 03 CE	18	JMP VTABZ	;exit thru VTABZ
CBE6:	19	*	
CBE6: CBEE	20	X.LF2 EQU *	
CBE6:CE FB 05	21	DEC OURCV	;back up to bottom
CBE9:C6 25	22	DEC CV	;and fall into scroll
CBEB:	23	*	
CBEB:A0 01	24	SCROLLUP LDY #1	;direction = up
CBED:8A	25	SCROLLIT TXA	;save X
CBEE:48	26	PHA	
CBEF:8C 7B 07	27	STY TEMP1	;save direction

CBF2:A5 21	28	LDA WNDWDTH	;get width of screen window
CBF4:48	29	PHA	;save original width
CBF5:2C 1F C0	30	BIT RD8OVID	;in 40 or 80 columns?
CBF8:10 1C CC16	31	BPL GETST1	;=>40, determine starting line
CBFA:8D 01 C0	32	STA SET80COL	;make sure this is enabled
CBFD:4A	33	LSR A	;divide by 2 for 80 column index
CBFE:AA	34	TAX	;and save
CBFF:A5 20	35	LDA WNDLFT	;test oddity of right edge
CC01:4A	36	LSR A	;by rotating low bit into carry
CC02:B8	37	CLV	;V=0 if left edge even
CC03:90 03 CC08	38	BCC CHKRT	;=>check right edge
CC05:2C 06 CB	39	BIT SEV1	;V=1 if left edge odd
CC08:2A	40	CHKRT ROL A	;restore WNDLFT
CC09:45 21	41	EOR WNDWDTH	;get oddity of right edge
CC0B:4A	42	LSR A	;C=1 if right edge even
CC0C:70 03 CC11	43	BVS GETST	;if odd left, don't DEY
CC0E:B0 01 CC11	44	BCS GETST	;if even right, don't DEY
CC10:CA	45	DEX	;if right edge odd, need one less
CC11:86 21	46	GETST STX WNDWDTH	;save window width
CC13:AD 1F C0	47	LDA RD8OVID	;N=1 if 80 columns
CC16:08	48	GETST1 PHP	;save N,Z,V
CC17:A6 22	49	LDX WNDTOP	;assume scroll from top
CC19:98	50	TYA	;up or down?
CC1A:D0 03 CC1F	51	BNE SETDBAS	;=>up
CC1C:A6 23	52	LDX WNDBTM	;down, start scrolling at bottom
CC1E:CA	53	DEX	;really need one less
CC1F:	54	*	
CC1F:8A	55	SETDBAS TXA	;get current line
CC20:20 03 CE	56	JSR VTABZ	;calculate base with window width
CC23:	57	*	
CC23:A5 28	58	SCRLIN LDA BASL	;current line is destination
CC25:85 2A	59	STA BAS2L	
CC27:A5 29	60	LDA BASH	
CC29:85 2B	61	STA BAS2H	
CC2B:	62	*	
CC2B:AD 7B 07	63	LDA TEMPl	;test direction
CC2E:F0 32 CC62	64	BEQ SCRLEDN	;=>do the downer
CC30:E8	65	INX	;do next line
CC31:E4 23	66	CPX WNDBTM	;done yet?
CC33:B0 32 CC67	67	BCS SCRLL3	;=>yup, all done
CC35:8A	68	SETSRC TXA	;set new line
CC36:20 03 CE	69	JSR VTABZ	;get base for new current line
CC39:A4 21	70	LDY WNDWDTH	;get width for scroll
CC3B:28	71	PLP	;get status for scroll
CC3C:08	72	PHP	;N=1 if 80 columns
CC3D:10 1E CC5D	73	BPL SKPRT	;=>only do 40 columns
CC3F:AD 55 C0	74	LDA TXTPAGE2	;scroll aux page first (even bytes)
CC42:98	75	TYA	;test Y
CC43:F0 07 CC4C	76	BEQ SCRLEFT	;if Y=0, only scroll one byte
CC45:B1 28	77	SCRLEVEN LDA (BASL),Y	
CC47:91 2A	78	STA (BAS2L),Y	
CC49:88	79	DEY	
CC4A:D0 F9 CC45	80	BNE SCRLEVEN	;do all but last even byte
CC4C:70 04 CC52	81	SCRLEFT BVS SKPLFT	;odd left edge, skip this byte

```

CC4E:B1 28      82      LDA  (BASL),Y
CC50:91 2A      83      STA  (BAS2L),Y
CC52:AD 54 CO   84 SKPLFT LDA  TXTPAGE1 ;now do main page (odd bytes)
CC55:A4 21      85      LDY  WNDWDTH ;restore width
CC57:B0 04      86      BCS  SKPRT ;even right edge, skip this byte
CC59:B1 28      87 SCRLDD LDA  (BASL),Y
CC5B:91 2A      88      STA  (BAS2L),Y
CC5D:88      89 SKPRT  DEY
CC5E:10 F9      90      BPL  SCRLDD
CC60:30 C1      91      BMI  SCRLIN ;=> always scroll next line
CC62:      92 *
CC62:CA      93 SCRLDN DEX ;do next line
CC63:E4 22      94      CPX  WNDTOP ;done yet
CC65:10 CE      95      BPL  SETSRC ;=>nope, not yet
CC67:      96 *
CC67:28      97 SCRL3 PLP ;pull status off stack
CC68:68      98      PLA  ;restore window width
CC69:85 21      99      STA  WNDWDTH
CC6B:20 96 CC   100     JSR  X.SUB ;clear current line
CC6E:20 FE CD   101     JSR  VTAB ;restore original cursor line
CC71:68      102     PLA  ;and X
CC72:AA      103     TAX
CC73:60      104     RTS ;done!!!
CC74:      105 *
CC74:      106 * EXECUTE CLR TO EOS:
CC74:      107 *
CC74:20 9A CC   108 X.VT JSR  X.GS ;CLEAR TO EOL
CC77:A5 25      109     LDA  CV ;SAVE CV
CC79:48      110     PHA
CC7A:10 06      111     BPL  X.VTNEXT ;DO NEXT LINE (ALWAYS TAKEN)
CC7C:20 03 CE   112 X.VTLOOP JSR VTABZ ;set base address
CC7F:20 96 CC   113     JSR  X.SUB ;CLEAR LINE
CC82:E6 25      114 X.VTNEXT INC CV
CC84:A5 25      115     LDA  CV
CC86:C5 23      116     CMP  WNDBTM ;OFF SCREEN?
CC88:90 F2      117     BCC  X.VTLOOP ;=>NO, KEEP GOING
CC8A:68      118     PLA  ;RESTORE
CC8B:85 25      119     STA  CV ; CV
CC8D:4C FE CD   120     JMP  VTAB ;return via VTAB (blech)
CC90:      121 *
CC90:      122 * EXECUTE CLEAR:
CC90:      123 *
CC90:      124 X.FF EQU  *
CC90:20 5F CB   125     JSR  X.EM ;HOME THE CURSOR
CC93:4C 74 CC   126     JMP  X.VT ;RETURN VIA CLREOS (UGH!)
CC96:      127 *
CC96:      128 * EXECUTE CLEAR LINE
CC96:      129 *
CC96:A0 00      130 X.SUB LDY #0 ;start at left
CC98:F0 03      131     BEQ  X.GSEOLZ ;and clear to end of line
CC9A:      132 *
CC9A:      133 * EXECUTE CLEAR TO EOL:
CC9A:      134 *
CC9A:AC 7B 05   135 X.GS LDY OURCH ;get CH

```

```

CC9D:A5 32      136 X.GSEOLZ LDA INVFLG      ;mask blank
CC9F:29 80      137          AND  #$80      ;with high bit of invflg
CCA1:09 20      138          ORA   #$20      ;make it a blank
CCA3:2C 1F CO    139          BIT   RD80VID    ;is it 80 columns?
CCA6:30 15 CCB0  140          BMI   CLR80      ;=>yes do quick clear
CCA8:91 28      141 CLR40   STA  (BASL),Y
CCAA:C8         142          INY
CCAB:C4 21      143          CPY   WNDWDTH
CCAD:90 F9 CCA8  144          BCC   CLR40
CCAF:60         145          RTS
CCB0:         146 *
CCB0:         147 * Clear right half of screen for 40 to 80
CCB0:         148 * screen conversion
CCB0:         149 *
CCB0:86 2A      150 CLRHALF STX  BAS2L      ;save X
CCB2:A2 D8      151          LDX  #$D8      ;set horizontal counter
CCB4:A0 14      152          LDY   #20
CCB6:A5 32      153          LDA  INVFLG    ;set (inverse) blank
CCB8:29 A0      154          AND  #$A0
CCBA:4C D5 CC    155          JMP   CLR2
CCBD:         156 *
CCBD:         157 * Clear to end of line for 80 columns
CCBD:         158 *
CCBD:86 2A      159 CLR80   STX  BAS2L      ;save X
CCBF:48         160          PHA          ;and blank
CCC0:98         161          TYA          ;get count for CH
CCC1:48         162          PHA          ;save for left edge check
CCC2:38         163          SEC          ;count=WNDWDTH-Y-1
CCC3:E5 21      164          SBC   WNDWDTH
CCC5:AA         165          TAX          ;save CH counter
CCC6:98         166          TYA          ;div CH by 2 for half pages
CCC7:4A         167          LSR   A
CCC8:A8         168          TAY
CCC9:68         169          PLA          ;restore original ch
CCCA:45 20      170          EOR   WNDLFT    ;get starting page
CCCC:6A         171          ROR   A
CCCD:B0 03 CCD2  172          BCS   CLR0
CCCF:10 01 CCD2  173          BPL   CLR0
CCD1:C8         174          INY          ;iff WNDLFT odd, starting byte odd
CCD2:68         175 CLR0   PLA          ;get blank
CCD3:B0 0B CCE0  176          BCS   CLR1    ;starting page is 1 (default)
CCD5:2C 55 CO    177 CLR2   BIT   TXTPAGE2   ;else do page 2
CCD8:91 28      178          STA  (BASL),Y
CCDA:2C 54 CO    179          BIT   TXTPAGE1   ;now do page 1
CCDD:E8         180          INX
CCDE:F0 06 CCE6  181          BEQ   CLR3      ;all done
CCE0:91 28      182 CLR1   STA  (BASL),Y
CCE2:C8         183          INY          ;forward 2 columns
CCE3:E8         184          INX          ;next ch
CCE4:D0 EF CCD5  185          BNE   CLR2      ;not done yet
CCE6:A6 2A      186 CLR3   LDX  BAS2L      ;restore X
CCE8:38         187          SEC          ;good exit condition
CCE9:60         188          RTS          ;and return
CCEA:         189 *

```

```

CCEA:          190 * EXECUTE '40COL MODE':
CCEA:          191 *
CCEA:          192 X.DC1 EQU *
CCEA:AD FB 04  193 LDA MODE ;don't convert if Pascal
CCED:30 4D
    CD3C 194 BMI X.DC1RTS ;=>it's Pascal
CCEF:20 31 CD  195 X.DC1A JSR SETTOP ;set top of window (0 or 20)
CCF2:2C 1F C0  196 BIT RD80VID ;are we in 80 columns?
CCF5:10 12 CD09 197 BPL X.DC1B ;=>no, no convert needed
CCF7:20 91 CD  198 JSR SCR84 ;else convert 80 to 40
CCFA:90 0D CD09 199 BCC X.DC1B ;=>always set new window
CCFC:          200 *
CCFC:          201 * Set 80 column mode
CCFC:          202 *
CCFC:          203 X.DC2 EQU *
CCFC:20 90 CA  204 JSR TESTCARD ;is there an 80 column card?
CCFF:D0 3B CD3C 205 BNE X.DC1RTS ;=>no, can't do this
CD01:2C 1F C0  206 BIT RD80VID ;are we in 40 columns?
CD04:30 03 CD09 207 BMI X.DC1B ;=>no, no convert needed
CD06:20 C4 CD  208 JSR SCR48 ;else convert 40 to 80
CD09:          209 *
CD09:AD 7B 05  210 X.DC1B LDA OURCH ;get cursor
CDOC:18        211 CLC ;since new window left = 0
CD0D:65 20     212 ADC WNDLFT ;NEWCH=OLDCH+OLDWNDLFT
CD0F:2C 1F C0  213 BIT RD80VID ;in 80 columns?
CD12:30 06 CD1A 214 BMI X.DC1C ;=>yes, CH is ok
CD14:C9 28     215 CMP #40 ;else if CH is too big,
CD16:90 02 CD1A 216 BCC X.DC1C ;set it to 39
CD18:A9 27     217 LDA #39
CD1A:8D 7B 05  218 X.DC1C STA OURCH ;save new CH
CD1D:85 24     219 STA CH
CD1F:A5 25     220 LDA CV ;base
CD21:20 BA CA  221 JSR BASCALC
CD24:2C 1F C0  222 BIT RD80VID ;in 80 columns?
CD27:10 05 CD2E 223 BPL D040 ;=>no, set forty column window
CD29:          224 *
CD29:20 71 CD  225 D080 JSR FULL80 ;set 80 column window
CD2C:F0 03 CD31 226 BEQ SETTOP ;=>always branch
CD2E:          227 *
CD2E:20 6D CD  228 D040 JSR FULL40 ;set 40 column window
CD31:A9 00     229 SETTOP LDA #0 ;assume normal window
CD33:2C 1A C0  230 BIT RDTEXT ;text or mixed?
CD36:30 02 CD3A 231 BMI D040A ;=>text, all ok
CD38:A9 14     232 LDA #20
CD3A:85 22     233 D040A STA WNDTOP ;set new top
CD3C:60        234 X.DC1RTS RTS
CD3D:          235 *
CD3D:          236 * EXECUTE MOUSE TEXT OFF
CD3D:          237 *
CD3D:AD FB 04  238 MOUSEOFF LDA MODE
CD40:09 01     239 ORA #M.MOUSE ;set mouse bit
CD42:D0 05 CD49 240 BNE SMOUSE ;to disable mouse chars
CD44:          241 *
CD44:          242 * EXECUTE MOUSE TEXT ON

```

```

CD44:          243 *
CD44:AD FB 04  244 MOUSEON LDA  MODE
CD47:29 FE     245          AND  #255-M.MOUSE ;clear mouse bit
CD49:8D FB 04  246 SMOUSE STA  MODE          ;to enable mouse chars
CD4C:60        247          RTS
CD4D:          248 *
CD4D:          249 * EXECUTE 'QUIT':
CD4D:          250 *
CD4D:          CD4D 251 X.NAK  EQU  *
CD4D:AD FB 04  252          LDA  MODE          ;ONLY VALID IN BASIC
CD50:30 1A    CD6C 253          BMI  SKRTS      ;ignore if pascal
CD52:20 2E CD  254          JSR  D040      ;force 40 column window
CD55:20 80 CD  255          JSR  QUIT      ;do stuff used by PR#0
CD58:20 64 CD  256          JSR  SETCOUT1   ;set output hook
CD5B:          257 *
CD5B:A9 FD     258 SETKEYIN LDA  #<KEYIN   ;set input hook
CD5D:85 39     259          STA  KSWH
CD5F:A9 1B     260          LDA  #>KEYIN
CD61:85 38     261          STA  KSWL
CD63:60        262          RTS
CD64:          263 *
CD64:A9 FD     264 SETCOUT1 LDA  #<COUT1   ;set output hook
CD66:85 37     265          STA  CSWH
CD68:A9 F0     266          LDA  #>COUT1
CD6A:85 36     267          STA  CSWL
CD6C:60        268 SKRTS   RTS
CD6D:          269 *
CD6D:          270 *****
CD6D:          271 * NAME      : FULL40
CD6D:          272 * FUNCTION: SET FULL 40COL WINDOW
CD6D:          273 * INPUT      : NONE
CD6D:          274 * OUTPUT    : WINDOW PARAMETERS, A=0
CD6D:          275 * VOLATILE: AC
CD6D:          276 *****
CD6D:          277 *
CD6D:          CD6D 278 FULL40 EQU  *
CD6D:A9 28     279          LDA  #40          ;set window width to 40
CD6F:D0 02    CD73 280          BNE  SAVWDTH   ;=>(always taken)
CD71:          281 *
CD71:          282 *****
CD71:          283 * NAME      : FULL80
CD71:          284 * FUNCTION: SET FULL 80COL WINDOW
CD71:          285 * INPUT      : NONE
CD71:          286 * OUTPUT    : WINDOW PARAMETERS, A=0
CD71:          287 * VOLATILE: AC
CD71:          288 *****
CD71:          289 *
CD71:A9 50     290 FULL80  LDA  #80          ;set full 80 column window
CD73:85 21     291 SAVWDTH STA  WNDWDTH
CD75:A9 18     292          LDA  #24
CD77:85 23     293          STA  WNDBTM
CD79:A9 00     294          LDA  #0
CD7B:85 22     295          STA  WNDTOP
CD7D:85 20     296          STA  WNDLFT

```

```

CD7F:60      297      RTS
CD80:        298 *
CD80:        299 * QUIT is used by PR#0 to turn off everything
CD80:        300 *
CD80:        CD80 301 QUIT EQU *
CD80:2C 1F C0 302 BIT RD80VID ;were we in 80 columns?
CD83:10 03 CD88 303 BPL QUIT2 ;=> not a chance
CD85:20 EF CC 304 JSR X.DC1A ;switch to 40 columns
CD88:8D 0E C0 305 QUIT2 STA CLRALTCHAR ;don't use lower case
CD8B:A9 FF 306 LDA #$FF ;DESTROY THE
CD8D:8D FB 04 307 STA MODE ; MODE BYTE
CD90:60      308 RTS
CD91:        309 *
CD91:        310 * SCR84 and SCR48 convert screens between 40 & 80 cols.
CD91:        311 * WNDTOP must be set up to indicate the last line to
CD91:        312 * be done. All registers are trashed.
CD91:        313 *
CD91:8A      314 SCR84 TXA ;save X
CD92:48      315 PHA
CD93:A2 17 316 LDX #23 ;start at bottom of screen
CD95:8D 01 C0 317 STA SET80COL ;allow page 2 access
CD98:8A      318 SCR1 TXA ;calc base for line
CD99:20 BA CA 319 JSR BASCALC
CD9C:A0 27 320 LDY #39 ;start at right of screen
CD9E:84 2A 321 SCR2 STY BAS2L ;save 40 index
CDA0:98      322 TYA ;div by 2 for 80 column index
CDA1:4A      323 LSR A
CDA2:B0 03 CDA7 324 BCS SCR3
CDA4:2C 55 C0 325 BIT TXTPAGE2 ;even column, do page 2
CDA7:A8      326 SCR3 TAY ;get 80 index
CDA8:B1 28 327 LDA (BASL),Y ;get 80 char
CDA A:2C 54 C0 328 BIT TXTPAGE1 ;restore pagel
CDAD:A4 2A 329 LDY BAS2L ;get 40 index
CDAF:91 28 330 STA (BASL),Y
CDB1:88      331 DEY
CDB2:10 EA CD9E 332 BPL SCR2 ;do next 40 byte
CDB4:CA      333 DEX ;do next line
CDB5:30 04 CDBB 334 BMI SCR4 ;=>done with setup
CDB7:E4 22 335 CPX WNDTOP ;at top yet?
CDB9:B0 DD CD98 336 BCS SCR1
CDBB:8D 00 C0 337 SCR4 STA CLR80COL ;clear 80STORE for 40 columns
CDBE:8D 0C C0 338 STA CLR80VID ;clear 80VID for 40 columns
CDC1:4C F8 CD 339 JMP SCRNRRET ;calc base, restore X, exit
CDC4:        340 *
CDC4:8A      341 SCR48 TXA ;save X
CDC5:48      342 PHA
CDC6:A2 17 343 LDX #23 ;start at bottom of screen
CDC8:8A      344 SCR5 TXA ;set base for current line
CDC9:20 BA CA 345 JSR BASCALC
CDCC:A0 00 346 LDY #0 ;start at left of screen
CDCE:8D 01 C0 347 STA SET80COL ;enable page2 store
CDD1:B1 28 348 SCR6 LDA (BASL),Y ;get 40 column char
CDD3:84 2A 349 SCR8 STY BAS2L ;save 40 column index
CDD5:48      350 PHA ;save char

```



```

CDD6:98          351      TYA          ;div 2 for 80 column index
CDD7:4A          352      LSR A
CDD8:B0 03 CDD 353      BCS SCR7      ;save on pagel
CDDA:8D 55 C0 354      STA TXTPAGE2
CDDD:A8          355 SCR7 TAY          ;get 80 column index
CDDE:68          356      PLA          ;now save character
CDDF:91 28      357      STA (BASL),Y
CDE1:8D 54 C0 358      STA TXTPAGE1    ;flip pagel
CDE4:A4 2A      359      LDY BAS2L      ;restore 40 column index
CDE6:C8          360      INY          ;move to the right
CDE7:C0 28      361      CPY #40        ;at right yet?
CDE9:90 E6 CDD1 362      BCC SCR6      ;=>no, do next column
CDEB:20 B0 CC 363      JSR CLRHALL      ;clear half of screen
CDEE:CA          364      DEX          ;else do next line of screen
CDEF:30 04 CDF5 365      BMI SCR9      ;=>done with top line
CDF1:E4 22      366      CPX WNDTOP     ;at top yet?
CDF3:B0 D3 CDC8 367      BCS SCR5
CDF5:8D 0D C0 368 SCR9 STA SET80VID    ;convert to 80 columns
CDF8:20 FE CD 369 SCRNRRT JSR VTAB     ;update base
CDFB:68          370      PLA          ;restore X
CDFC:AA          371      TAX
CDFD:60          372      RTS
CDFE:           373 *
CDFE:A5 25      374 VTAB LDA CV        ;get 80 column CV
CE00:8D FB 05 375      STA OURCV      ;copy to OURCV
CE03:20 BA CA 376 VTABZ JSR BASCALC    ;calc base address
CE06:A5 20      377      LDA WNDLFT    ;and add window left to it
CE08:2C 1F C0 378      BIT RD80VID    ;is it 80 columns?
CE0B:10 01 CEOE 379      BPL VTAB40    ;window width ok
CE0D:4A          380      LSR A        ;else divide width by 2
CE0E:18          381 VTAB40 CLC        ;prepare to add
CE0F:65 28      382      ADC BASL      ;add in window left
CE11:85 28      383      STA BASL      ;and update base
CE13:60          384 VTABX RTS        ;and exit
CE14:           29      INCLUDE SUBS3
CE14:C9 E1      1 UPSHFT CMP #E1      ;is it lowercase?
CE16:90 06 CE1E 2      BCC UPSHFT2    ;=>nope
CE18:C9 FB      3      CMP #F8B      ;lowercase?
CE1A:B0 02 CE1E 4      BCS UPSHFT2    ;=>nope
CE1C:29 DF      5      AND #DF        ;else upshift
CE1E:60          6 UPSHFT2 RTS
CE1F:           7 *
CE1F:           8 *****
CE1F:           9 * NAME : INVERT
CE1F:          10 * FUNCTION: INVERT CHAR AT CH/CV
CE1F:          11 * : Unless Pascal and M.CURSOR=1
CE1F:          12 * INPUT : NOTHING
CE1F:          13 * OUTPUT : CHAR AT CH/CV INVERTED
CE1F:          14 * VOLATILE: NOTHING
CE1F:          15 * CALLS : PICK, STORCHAR
CE1F:          16 *****
CE1F:          17 *
CE1F:AD FB 04 18 PASINV LDA MODE      ;check pascal cursor flag
CE22:29 10 19      AND #M.CURSOR ;before displaying cursor

```

```

CE24:DO 11 CE37 20          BNE  INVX      ;=>cursor off, don't invert
CE26:48          21  INVERT PHA          ;save AC
CE27:98          22          TYA          ; AND Y
CE28:48          23          PHA
CE29:AC 7B 05    24          LDY  OURCH    ;GET CH
CE2C:20 44 CE    25          JSR  PICK     ;GET CHARACTER
CE2F:49 80       26          EOR  #$80     ;FLIP INVERSE/NORMAL
CE31:20 70 CE    27          JSR  STORIT    ; ONTO SCREEN
CE34:68          28          PLA          ;RESTORE Y
CE35:A8          29          TAY          ; AND AC
CE36:68          30          PLA
CE37:60          31  INVX   RTS
CE38:           32  *****
CE38:           33  * NAME    : STORCHAR
CE38:           34  * FUNCTION: STORE A CHAR ON SCREEN
CE38:           35  * INPUT   : AC=CHAR
CE38:           36  *          : Y=CH POSITION
CE38:           37  * OUTPUT  : CHAR ON SCREEN
CE38:           38  * VOLATILE: NOTHING
CE38:           39  * CALLS   : SCREENIT
CE38:           40  *****
CE38:           41  *
CE38:           42  STORCHAR EQU *
CE38:48 CE38     43          PHA          ;SAVE AC
CE39:24 32       44          BIT  INVFLG    ;NORMAL OR INVERSE?
CE3B:30 02 CE3F  45          BMI  STOR2     ;=>NORMAL
CE3D:29 7F       46          AND  #$7F     ;inverse it
CE3F: CE3F      47  STOR2  EQU  *
CE3F:20 70 CE    48          JSR  STORIT    ;=>do it!!
CE42:68          49          PLA          ;RESTORE AC
CE43:60          50  SEV    RTS
CE44:           51  *****
CE44:           52  * NAME    : PICK
CE44:           53  * FUNCTION: GET A CHAR FROM SCREEN
CE44:           54  * INPUT   : Y=CH POSITION
CE44:           55  * OUTPUT  : AC=CHARACTER
CE44:           56  * VOLATILE: NOTHING
CE44:           57  * CALLS   : SCREENIT
CE44:           58  *****
CE44:           59  *
CE44:B1 28       60  PICK   LDA  (BASL),Y  ;get 40 column character
CE46:2C 1F C0    61          BIT  RD80VID   ;80 columns?
CE49:10 19 CE64  62          BPL  PICK3     ;=>no, do text shift
CE4B:8D 01 C0    63          STA  SET80COL   ;force 80STORE for 80 columns
CE4E:84 2A       64          STY  BAS2L     ;temp store for position
CE50:98          65          TYA          ;divide CH by two
CE51:45 20       66          EOR  WNDLFT    ;C=1 if char in main RAM
CE53:6A          67          ROR  A         ;get low bit into carry
CE54:B0 04 CE5A  68          BCS  PICK1     ;=>store in main memory
CE56:AD 55 C0    69          LDA  TXTPAGE2   ;else switch in page 2
CE59:C8          70          INY          ;for odd left, aux bytes
CE5A:98          71  PICK1  TYA          ;divide position by 2
CE5B:4A          72          LSR  A         ;and use carry as
CE5C:A8          73          TAY          ;page indicator

```

CE5D:B1 28	74 PICK2	LDA (BASL),Y	;get that char
CE5F:2C 54 C0	75	BIT TXTPAGE1	;flip to page 1
CE62:A4 2A	76	LDY BAS2L	
CE64:2C 1E C0	77 PICK3	BIT ALTCHARSET	;only allow mouse text
CE67:10 06 CE6F	78	BPL PICK4	;if alternate character set
CE69:C9 20	79	CMP #\$20	
CE6B:B0 02 CE6F	80	BCS PICK4	
CE6D:09 40	81	ORA #\$40	
CE6F:60	82 PICK4	RTS	
CE70:	83 *		
CE70:	84	*****	
CE70:	85 *	NAME : STORIT	
CE70:	86 *	FUNCTION: STORE CHAR	
CE70:	87 *	INPUT : AC=char for store	
CE70:	88 *	: Z=high bit of char	
CE70:	89 *	: Y=CH POSITION	
CE70:	90 *	OUTPUT : AC=CHAR (PICK)	
CE70:	91 *	VOLATILE: NOTHING	
CE70:	92 *	CALLS : NOTHING	
CE70:	93	*****	
CE70:	94 *		
CE70:48	95 STORIT	PHA	;save char
CE71:29 FF	96	AND #\$FF	;if high bit set...
CE73:30 16 CE8B	97	BMI STORE1	;=>not mouse text
CE75:AD FB 04	98	LDA MODE	;is mouse text enabled?
CE78:6A	99	ROR A	;use carry as flag
CE79:68	100	PLA	;and restore char
CE7A:48	101	PHA	;need to save it too
CE7B:90 0E CE8B	102	BCC STORE1	
CE7D:2C 1E C0	103	BIT ALTCHARSET	;only do mouse text if
CE80:10 09 CE8B	104	BPL STORE1	;alt char set switched in
CE82:49 40	105	EOR #\$40	;do mouse shift
CE84:2C AC CE	106	BIT HEX60	;is it in proper range?
CE87:F0 02 CE8B	107	BEQ STORE1	;=>yes, leave it
CE89:49 40	108	EOR #\$40	;else shift it back
CE8B:	109 *		
CE8B:2C 1F C0	110 STORE1	BIT RD8OVID	;80 columns?
CE8E:10 1D CEAD	111	BPL STOR40	;=>no, 40 columns
CE90:8D 01 C0	112	STA SET80COL	;force 80STORE for 80 columns
CE93:48	113	PHA	;save shifted character
CE94:84 2A	114	STY BAS2L	;temp storage
CE96:98	115	TYA	;get position
CE97:45 20	116	EOR WNDLFT	;C=1 if char in main RAM
CE99:4A	117	LSR A	
CE9A:B0 04 CEAO	118	BCS STORE2	;=>yes, main RAM
CE9C:AD 55 C0	119	LDA TXTPAGE2	;else flip in main RAM
CE9F:C8	120	INY	;do this for odd left bytes
CEA0:98	121 STORE2	TYA	;get position
CEA1:4A	122	LSR A	;and divide it by 2
CEA2:A8	123	TAY	
CEA3:68	124 STORIT2	PLA	;restore acc
CEA4:91 28	125	STA (BASL),Y	;save to screen
CEA6:AD 54 C0	126	LDA TXTPAGE1	;flip to page 1
CEA9:A4 2A	127	LDY BAS2L	

```

CEAB:68      128      PLA      ;restore true Acc
CEAC:60      129 HEX60  RTS      ;and exit
CEAD:        130 *
CEAD:91 28   131 STOR40 STA (BASL),Y ;quick 40 column store
CEAF:68      132      PLA      ;restore real char
CEB0:60      133      RTS
CEB1:        134 *****
CEB1:        135 * NAME      : ESCON
CEB1:        136 * FUNCTION: TURN ON 'ESCAPE' CURSOR
CEB1:        137 * INPUT      : NONE
CEB1:        138 * OUTPUT     : 'CHAR'=ORIGINAL CHAR
CEB1:        139 * VOLATELE: NOTHING
CEB1:        140 * CALLS      : PICK,STORCHAR
CEB1:        141 *****
CEB1:        142 *
CEB1:        CEB1 143 ESCON EQU *
CEB1:48      144      PHA      ;SAVE AC
CEB2:98      145      TYA      ; AND Y
CEB3:48      146      PHA
CEB4:AC 7B 05 147      LDY OURCH ;GET CH
CEB7:20 44 CE 148      JSR PICK ;GET ORIGINAL CHARACTER
CEBA:8D 7B 06 149      STA CHAR ; AND REMEMBER FOR ESCOFF
CEBD:29 80    150      AND #$80 ;SAVE NORMAL/INVERSE BIT
CEBF:49 AB    151      EOR #$AB ;MAKE IT AN INVERSE '+'
CEC1:4C CD CE 152      JMP ECRET ;RETURN VIA SIMILAR CODE
CEC4:        153 *****
CEC4:        154 * NAME      : ESCOFF
CEC4:        155 * FUNCTION: TURN OFF 'ESCAPE' CURSOR
CEC4:        156 * INPUT      : 'CHAR'=ORIGINAL CHAR
CEC4:        157 * OUTPUT     : NONE
CEC4:        158 * VOLATILE: NOTHING
CEC4:        159 * CALLS      : STORCHAR
CEC4:        160 *****
CEC4:        161 *
CEC4:        CEC4 162 ESCOFF EQU *
CEC4:48      163      PHA      ;SAVE AC
CEC5:98      164      TYA      ; AND Y
CEC6:48      165      PHA
CEC7:AC 7B 05 166      LDY OURCH ;GET CH
CECA:AD 7B 06 167      LDA CHAR ;GET ORIGINAL CHARACTER
CECD:        CECD 168 ECRET EQU * ;USED BY ESCON
CECD:20 70 CE 169      JSR STORIT ; EXACTLY AS IT WAS
CED0:68      170      PLA      ;RESTORE Y
CED1:A8      171      TAY
CED2:68      172      PLA      ; AND AC
CED3:60      173      RTS
CED4:        174 *****
CED4:        175 * NAME      : PSETUP
CED4:        176 * FUNCTION: SETUP ZP FOR PASCAL
CED4:        177 * INPUT      : NONE
CED4:        178 * OUTPUT     : NONE
CED4:        179 * VOLATILE: AC
CED4:        180 * CALLS      : NOTHING
CED4:        181 *****

```

```

CED4:          182 *
CED4:          CED4 183 PSETUP EQU *
CED4:20 71 CD    184          JSR FULL80      ;SET FULL 80COL WINDOW
CED7:A9 FF      185 IS80   LDA #255
CED9:85 32      186          STA INVFLG      ;ASSUME NORMAL MODE
CEDB:          187 *
CEDB:AD FB 04   188          LDA MODE
CEDE:29 04      189          AND #M.VMODE
CEE0:F0 02     CEE4 190          BEQ PSETUPRET ;=>IT'S NORMAL
CEE2:46 32      191          LSR INVFLG      ;MAKE IT INVERSE
CEE4:          192 *
CEE4:          CEE4 193 PSETUPRET EQU *
CEE4:AD 7B 07   194          LDA OLDBASL      ;SET UP BASE ADDRESS
CEE7:85 28      195          STA BASL
CEE9:AD FB 07   196          LDA OLDBASH
CEEC:85 29      197          STA BASH
EEEE:AD FB 05   198          LDA OURCV      ;get user's cursor vertical
CEF1:85 25      199          STA CV        ;and set it up
CEF3:60         200          RTS
CEF4:          201 *****
CEF4:          202 *
CEF4:          203 * COPYROM is called when the video firmware is
CEF4:          204 * initialized. If the language card is switched
CEF4:          205 * in for reading, it copies the F8 ROM to the
CEF4:          206 * language card and restores the state of the
CEF4:          207 * language card.
CEF4:          208 *
CEF4:2C 12 CO   209 COPYROM BIT RDLGRAM      ;is the LC switched in?
CEF7:10 3D     CF36 210          BPL ROMOK      ;=>no, do nothing
CEF9:A9 06      211          LDA #GOODF8      ;yes, check $F8 RAM
CEFB:CD B3 FB   212          CMP F8VERSION ;does it match?
CEFE:F0 36     CF36 213          BEQ ROMOK      ;=> assum ROM is there
CF00:A2 03      214          LDX #3        ;indicate bank 2, RAM write enabled
CF02:2C 11 CO   215          BIT RDLCBNK2      ;is it bank 2?
CF05:30 02     CF09 216          BMI BANK2      ;=>yes, we were right
CF07:A2 0B      217          LDX #$B        ;no, bank 1, RAM write enabled
CF09:8D B3 FB   218 BANK2 STA F8VERSION ;write to see if LC is
CF0C:2C 80 CO   219          BIT $C080      ;write protected (read RAM)
CF0F:AD B3 FB   220          LDA F8VERSION ;did it change?
CF12:C9 06      221          CMP #GOODF8
CF14:F0 01     CF17 222          BEQ WRTENBL    ;=>yes, write enabled
CF16:E8         223          INX            ;else indicate write protect
CF17:2C 81 CO   224 WRTENBL BIT $C081      ;read ROM, write RAM
CF1A:2C 81 CO   225          BIT $C081      ;twice is nice
CF1D:A0 00      226          LDY #0        ;now copy ROM to RAM
CF1F:A9 F8      227          LDA #$F8
CF21:85 37      228          STA CSWH      ;hooks set later
CF23:84 36      229          STY CSWL
CF25:B1 36      230 COPYROM2 LDA (CSWL),Y    ;get a byte
CF27:91 36      231          STA (CSWL),Y    ;and move it
CF29:C8         232          INY
CF2A:D0 F9     CF25 233          BNE COPYROM2
CF2C:E6 37      234          INC CSWH      ;next page
CF2E:D0 F5     CF25 235          BNE COPYROM2 ;finish copy
CF30:BD 80 CO   236          LDA $C080,x    ;read RAM
CF33:BD 80 CO   237          LDA $C080,x
CF36:60         238 ROMOK   RTS            ;done with ROM copy

```

```

0000:      0000  1 TEST    EQU  0

0000:      0000  2          LST  On,A,V
0000:      0001  3 IRQTEST EQU  1
0000:      0000  4          MSB  ON          ;SET THEM HIBITS
0000:      0000  5          DO   TEST
S          6 F8ORG    EQU  $1800
S          7 IOADR    EQU  $2000      ;For setting PR# hooks
S          8 C1ORG    EQU  $2100
S          9 C3ORG    EQU  $2300
S         10 C8ORG    EQU  $2800
0000:      0000  11         ELSE
0000:      F800  12 F8ORG    EQU  $F800
0000:      C100  13 C1ORG    EQU  $C100
0000:      C300  14 C3ORG    EQU  $C300
0000:      C800  15 C8ORG    EQU  $C800
0000:      0000  16         FIN

0000:      0000  2 *****
0000:      0000  3 *
0000:      0000  4 * APPLE II
0000:      0000  5 * MONITOR II
0000:      0000  6 *
0000:      0000  7 * COPYRIGHT 1978, 1981, 1984 BY
0000:      0000  8 * APPLE COMPUTER, INC.
0000:      0000  9 *
0000:      0000 10 * ALL RIGHTS RESERVED
0000:      0000 11 *
0000:      0000 12 * S. WOZNIAK          1977
0000:      0000 13 * A. BAUM            1977
0000:      0000 14 * JOHN A              NOV 1978
0000:      0000 15 * R. AURICCHIO        SEP 1981
0000:      0000 16 * E. BEERNINK         1984
0000:      0000 17 *
0000:      0001 18 APPLE2F EQU  1          ;COND ASSM/RAA0981
0000:      0000 19 *
0000:      0000 20 *****
F800:      F800 21          ORG  F8ORG
F800:      2000 22          OBJ  $2000
F800:      0000 23 *****
F800:      0000 24 *
F800:      0000 25 * Zero Page Equates
F800:      0000 26 *
F800:      0000 27 LOC0     EQU  $00      ;vector for autost from disk
F800:      0001 28 LOC1     EQU  $01
F800:      00 20 29 WNDLFT   EQU  $20      ;left edge of text window
F800:      00 21 30 WNDWDTH  EQU  $21      ;width of text window
F800:      00 22 31 WNDTOP   EQU  $22      ;top of text window
F800:      00 23 32 WNDBTM   EQU  $23      ;bottom+1 of text window
F800:      00 24 33 CH        EQU  $24      ;cursor horizontal position
F800:      00 25 34 CV        EQU  $25      ;cursor vertical position
F800:      00 26 35 GBASL    EQU  $26      ;lo-res graphics base addr.
F800:      00 27 36 GBASH    EQU  $27
F800:      00 28 37 BASL     EQU  $28      ;text base address

```

F800:	0029	38	BASH	EQU	\$29	
F800:	002A	39	BAS2L	EQU	\$2A	;temp base for scrolling
F800:	002B	40	BAS2H	EQU	\$2B	
F800:	002C	41	H2	EQU	\$2C	;temp for lo-res graphics
F800:	002C	42	LMNEM	EQU	\$2C	;temp for mnemonic decoding
F800:	002D	43	V2	EQU	\$2D	;temp for lo-res graphics
F800:	002D	44	RMNEM	EQU	\$2D	;temp for mnemonic decoding
F800:	002E	45	MASK	EQU	\$2E	;color mask for lo-res gr.
F800:	002E	46	CHKSUM	EQU	\$2E	;temp for opcode decode
F800:	002E	47	FORMAT	EQU	\$2E	;temp for opcode decode
F800:	002F	48	LASTIN	EQU	\$2F	;temp for tape read csum
F800:	002F	49	LENGTH	EQU	\$2F	;temp for opcode decode
F800:	0030	50	COLOR	EQU	\$30	;color for lo-res graphics
F800:	0031	51	MODE	EQU	\$31	;Monitor mode
F800:	0032	52	INVFLG	EQU	\$32	;normal/inverse(/flash)
F800:	0033	53	PROMPT	EQU	\$33	;prompt character
F800:	0034	54	YSAV	EQU	\$34	;position in Monitor command
F800:	0035	55	YSAV1	EQU	\$35	;temp for Y register
F800:	0036	56	CSWL	EQU	\$36	;character output hook
F800:	0037	57	CSWH	EQU	\$37	
F800:	0038	58	KSWL	EQU	\$38	;character input hook
F800:	0039	59	KSWH	EQU	\$39	
F800:	003A	60	PCL	EQU	\$3A	;temp for program counter
F800:	003B	61	PCH	EQU	\$3B	
F800:	003C	62	A1L	EQU	\$3C	;A1-A5 are Monitor temps
F800:	003D	63	A1H	EQU	\$3D	
F800:	003E	64	A2L	EQU	\$3E	
F800:	003F	65	A2H	EQU	\$3F	
F800:	0040	66	A3L	EQU	\$40	
F800:	0041	67	A3H	EQU	\$41	
F800:	0042	68	A4L	EQU	\$42	
F800:	0043	69	A4H	EQU	\$43	
F800:	0044	70	A5L	EQU	\$44	
F800:	0044	71	MACSTAT	EQU	\$44	;machine state for break
F800:	0045	72	A5H	EQU	\$45	
F800:	0045	73	ACC	EQU	\$45	;Acc after break (destroys A5H)
F800:	0046	74	XREG	EQU	\$46	;X reg after break
F800:	0047	75	YREG	EQU	\$47	;Y reg after break
F800:	0048	76	STATUS	EQU	\$48	;P reg after break
F800:	0049	77	SPNT	EQU	\$49	;SP after break
F800:	004E	78	RNDL	EQU	\$4E	;random counter low
F800:	004F	79	RNDH	EQU	\$4F	;random counter high
F800:		80	*			
F800:	0095	81	PICK	EQU	\$95	;CONTROL-U character
F800:		82	*			
F800:	0200	83	IN	EQU	\$0200	;input buffer for GETLN
F800:		84	*			
F800:		85	* Page 3 vectors			
F800:		86	*			
F800:	03F0	87	BRKV	EQU	\$03F0	;vectors here after break
F800:	03F2	88	SOFTEV	EQU	\$03F2	;vector for warm start
F800:	03F4	89	PWREDUP	EQU	\$03F4	;THIS MUST = EOR #\$A5 OF SOFTEV+1
F800:	03F5	90	AMPERV	EQU	\$03F5	;APPLESOFT & EXIT VECTOR
F800:	03F8	91	USRADR	EQU	\$03F8	;Applesoft USR function vector


```

F800:      03FB  92 NMI      EQU  $03FB      ;NMI vector
F800:      03FE  93 IRQLOC  EQU  $03FE      ;Maskable interrupt vector
F800:      94 *
F800:      0400  95 LINE1   EQU  $0400      ;first line of text screen
F800:      07F8  96 MSLOT   EQU  $07F8      ;current user of $C8 space
F800:      97 *
F800:      0000  98          DO    TEST
F800:      99          ELSE
F800:      C000  100 IOADR   EQU  $C000
F800:      101          FIN
F800:      102 *
F800:      C000  103 KBD     EQU  $C000
F800:      C006  104 SLOTCXROM EQU  $C006      ;enable slots 1-7
F800:      C007  105 INTXCXROM EQU  $C007      ;swap out slots for firmware
F800:      C010  106 KBDSTRB EQU  $C010
F800:      C01F  107 RD80VID EQU  $C01F
F800:      C020  108 TAPEOUT EQU  $C020
F800:      C030  109 SPKR    EQU  $C030
F800:      C050  110 TXTCLR  EQU  $C050
F800:      C05 1  111 TXTSET  EQU  $C051
F800:      C05 2  112 MIXCLR  EQU  $C052
F800:      C05 3  113 MIXSET  EQU  $C053
F800:      C05 4  114 LOWSCR  EQU  $C054
F800:      C05 5  115 HISCR  EQU  $C055
F800:      C05 6  116 LORES  EQU  $C056
F800:      C05 7  117 HIRES  EQU  $C057
F800:      C05 8  118 SETANO  EQU  $C058
F800:      C05 9  119 CLRANO  EQU  $C059
F800:      C05 A  120 SETAN1  EQU  $C05A
F800:      C05 B  121 CLRAN1  EQU  $C05B
F800:      C05 C  122 SETAN2  EQU  $C05C
F800:      C05 D  123 CLRAN2  EQU  $C05D
F800:      C05 E  124 SETAN3  EQU  $C05E
F800:      C05 F  125 CLRAN3  EQU  $C05F
F800:      C06 0  126 TAPEIN  EQU  $C060
F800:      C06 4  127 PADDL0  EQU  $C064
F800:      C07 0  128 PTRIG   EQU  $C070
F800:      129 *
F800:      C3F A  130 IRQ     EQU  C30RG+$FA ;IRQ entry in $C3 page
F800:      C47 C  131 IRQFIX  EQU  C30RG+$17C ;Restore state at IRQ
F800:      132 *
F800:      C56 7  133 XHEADER EQU  C30RG+$267
F800:      C5D 1  134 XREAD   EQU  C30RG+$2D1
F800:      C5A A  135 WRITE2  EQU  C30RG+$2AA
F800:      136 *
F800:      CFFF  137 CLRROM  EQU  $CFFF
F800:      E0C 0  138 BASIC  EQU  $E000
F800:      E00 3  139 BASIC2  EQU  $E003
F800:      140 *
F800:4A      141 PLOT    LSR  A      ;Y-COORD/2
F801:08      142      PHP          ;SAVE LSB IN CARRY
F802:20 47 F8 143      JSR  GBASCALC ;CALC BASE ADR IN GBASL,H
F805:28      144      PLP          ;RESTORE LSB FROM CARRY
F806:A9 0F      145      LDA  #$0F    ;MASK $0F IF EVEN

```



```

F808:90 02   F80C 146      BCC RTMASK
F80A:69 E0      147      ADC #$E0      ;MASK $FO IF ODD
F80C:85 2E      148 RTMASK STA MASK
F80E:B1 26      149 PLOT1 LDA (GBASL),Y ;DATA
F810:45 30      150      EOR COLOR      ; XOR COLOR
F812:25 2E      151      AND MASK      ; AND MASK
F814:51 26      152      EOR (GBASL),Y ; XOR DATA
F816:91 26      153      STA (GBASL),Y ; TO DATA
F818:60      154      RTS
F819:      155 *
F819:20 00 F8    156 HLINE JSR PLOT      ;PLOT SQUARE
F81C:C4 2C      157 HLINE1 CPY H2      ;DONE?
F81E:B0 11   F831 158      BCS RTS1      ; YES, RETURN
F820:C8      159      INY      ; NO, INCR INDEX (X-COORD)
F821:20 0E F8    160      JSR PLOT1      ;PLOT NEXT SQUARE
F824:90 F6   F81C 161      BCC HLINE1      ;ALWAYS TAKEN
F826:69 01      162 VLINEZ ADC #$01      ;NEXT Y-COORD
F828:48      163 VLINE PHA      ; SAVE ON STACK
F829:20 00 F8    164      JSR PLOT      ; PLOT SQUARE
F82C:68      165      PLA
F82D:C5 2D      166      CMP V2      ;DONE?
F82F:90 F5   F826 167      BCC VLINEZ      ; NO, LOOP.
F831:60      168 RTS1 RTS
F832:      169 *
F832:A0 2F      170 CLRSCR LDY #$2F      ;MAX Y, FULL SCRN CLR
F834:D0 02   F838 171      BNE CLRSC2      ;ALWAYS TAKEN
F836:A0 27      172 CLRTOP LDY #$27      ;MAX Y, TOP SCRN CLR
F838:84 2D      173 CLRSC2 STY V2      ;STORE AS BOTTOM COORD
F83A:      174 ;      FOR VLINE CALLS
F83A:A0 27      175      LDY #$27      ;RIGHTMOST X-COORD (COLUMN)
F83C:A9 00      176 CLRSC3 LDA #$00      ;TOP COORD FOR VLINE CALLS
F83E:85 30      177      STA COLOR      ;CLEAR COLOR (BLACK)
F840:20 28 F8    178      JSR VLINE      ;DRAW VLINE
F843:88      179      DEY      ;NEXT LEFTMOST X-COORD
F844:10 F6   F83C 180      BPL CLRSC3      ;LOOP UNTIL DONE.
F846:60      181      RTS
F847:      182 *
F847:48      183 GBASCALC PHA      ;FOR INPUT 00DEF GH
F848:4A      184      LSR A
F849:29 03      185      AND #$03
F84B:09 04      186      ORA #$04      ;GENERATE GBASH=000001FG
F84D:85 27      187      STA GBASH
F84F:68      188      PLA      ;AND GBASL=HDEDE000
F850:29 18      189      AND #$18
F852:90 02   F856 190      BCC GBCALC
F854:69 7F      191      ADC #$7F
F856:85 26      192 GBCALC STA GBASL
F858:0A      193      ASL A
F859:0A      194      ASL A
F85A:05 26      195      ORA GBASL
F85C:85 26      196      STA GBASL
F85E:60      197      RTS
F85F:      198 *
F85F:A5 30      199 NXTCOL LDA COLOR      ;INCREMENT COLOR BY 3

```

F861:18	200	CLC		
F862:69 03	201	ADC	#\$03	
F864:29 0F	202	SETCOL	AND #\$0F	;SETS COLOR=17*A MOD 16
F866:85 30	203	STA	COLOR	
F868:0A	204	ASL	A	;BOTH HALF BYTES OF COLOR EQUAL
F869:0A	205	ASL	A	
F86A:0A	206	ASL	A	
F86B:0A	207	ASL	A	
F86C:05 30	208	ORA	COLOR	
F86E:85 30	209	STA	COLOR	
F870:60	210	RTS		
F871:	211	*		
F871:4A	212	SCRN	LSR A	;READ SCREEN Y-COORD/2
F872:08	213	PHP		;SAVE LSB (CARRY)
F873:20 47 F8	214	JSR	GBASCALC	;CALC BASE ADDRESS
F876:B1 26	215	LDA	(GBASL),Y	;GET BYTE
F878:28	216	PLP		;RESTORE LSB FROM CARRY
F879:90 04 F87 F	217	SCRN2	BCC RTMSKZ	;IF EVEN, USE LO H
F87B:4A	218	LSR	A	
F87C:4A	219	LSR	A	
F87D:4A	220	LSR	A	;SHIFT HIGH HALF BYTE DOWN
F87E:4A	221	LSR	A	
F87F:29 0F	222	RTMSKZ	AND #\$0F	;MASK 4-BITS
F881:60	223	RTS		
F882:	224	*		
F882:A6 3A	225	INSDS1	LDX PCL	;PRINT PCL,H
F884:A4 3B	226	LDY	PCH	
F886:20 96 FD	227	JSR	PRYX2	
F889:20 48 F9	228	JSR	PRBLNK	;FOLLOWED BY A BLANK
F88C:A1 3A	229	LDA	(PCL,X)	;GET OPCODE
F88E:A8	230	INSDS2	TAY	
F88F:4A	231	LSR	A	;EVEN/ODD TEST
F890:90 09 F89B	232	BCC	IEVEN	
F892:6A	233	ROR	A	;BIT 1 TEST
F893:B0 10 F8A5	234	BCS	ERR	;XXXXXX11 INVALID OP
F895:C9 A2	235	CMP	#\$A2	
F897:F0 0C F8A5	236	BEQ	ERR	;OPCODE \$89 INVALID
F899:29 87	237	AND	#\$87	;MASK BITS
F89B:4A	238	IEVEN	LSR A	;LSB INTO CARRY FOR L/R TEST
F89C:AA	239	TAX		
F89D:BD 62 F9	240	LDA	FMT1,X	;GET FORMAT INDEX BYTE
F8A0:20 79 F8	241	JSR	SCRN2	;R/L H-BYTE ON CARRY
F8A3:D0 04 F8A9	242	BNE	GETFMT	
F8A5:A0 80	243	ERR	LDY #\$80	;SUBSTITUTE \$80 FOR INVALID OPS
F8A7:A9 00	244	LDA	#\$00	;SET PRINT FORMAT INDEX TO 0
F8A9:AA	245	GETFMT	TAX	
F8AA:BD A6 F9	246	LDA	FMT2,X	;INDEX INTO PRINT FORMAT TABLE
F8AD:85 2E	247	STA	FORMAT	;SAVE FOR ADR FIELD FORMATTING
F8AF:	248	; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)		
F8AF:	249	*		
F8AF:	250	* Move code to C1-C2 because the code		
F8AF:	251	* that tests for ROM in slot 3 must be in		
F8AF:	252	* the F8 ROM.		
F8AF:	253	*		

F8AF:AA	254	TAX	;save ACC in X
F8B0:84 2A	255	STY BAS2L	;and Y in scrolling temp
F8B2:A0 10	256	LDY #\$10	;call = finish mnemonics
F8B4:4C B4 FB	257	JMP GOTOCX	;off to C100
F8B7:	258 *		
F8B7:	259 *	* Test slot 3 for a card containing ROM.	
F8B7:	260 *	* If there is one, we'll not switch in our internal	
F8B7:	261 *	* slot 3 firmware (for 80 columns).	
F8B7:	262 *	* On entry Y has a high value like \$F2, so the	
F8B7:	263 *	* ROM/bus is read a bunch of times	
F8B7:	264 *		
F8B7:8D 06 C0	265	TSTROM STA SLOTXROM	;swap in slots
F8BA:A2 02	266	TSTROM0 LDX #2	;check 2 ID bytes
F8BC:BD 05 C3	267	TSTROM1 LDA \$C305,X	;at C305 and \$C307
F8BF:DD 9C FC	268	CMP CLREOL,X	;with two bytes that are same
F8C2:D0 07 F8CB	269	BNE XTST	
F8C4:CA	270	DEX	;check next ID byte
F8C5:CA	271	DEX	
F8C6:10 F4 F8BC	272	BPL TSTROM1	
F8C8:88	273	DEY	
F8C9:D0 EF F8BA	274	BNE TSTROM0	;if ROM ok, exit with BEQ
F8CB:8D 07 C0	275	XTST STA INTCXROM	;swap internal ROM
F8CE:60	276	RTS	;and return there
F8CF:	277 *		
F8CF:EA	278	NOP	;line things up
F8D0:	279 *		
F8D0:20 82 F8	280	INSTDSP JSR INSDS1	;GEN FMT, LEN BYTES
F8D3:48	281	PHA	;SAVE MNEMONIC TABLE INDEX
F8D4:B1 3A	282	PRNTOP LDA (PCL),Y	
F8D6:20 DA FD	283	JSR PRBYTE	
F8D9:A2 01	284	LDX #\$01	;PRINT 2 BLANKS
F8DB:20 4A F9	285	PRNTBL JSR PRBL2	
F8DE:C4 2F	286	CPY LENGTH	;PRINT INST (1-3 BYTES)
F8E0:C8	287	INY	;IN A 12 CHR FIELD
F8E1:90 F1 F8D4	288	BCC PRNTOP	
F8E3:A2 03	289	LDX #\$03	;CHAR COUNT FOR MNEMONIC INDEX
F8E5:C0 04	290	CPY #\$04	
F8E7:90 F2 F8DB	291	BCC PRNTBL	
F8E9:68	292	PLA	;RECOVER MNEMONIC INDEX
F8EA:A8	293	TAY	
F8EB:B9 C0 F9	294	LDA MNEML,Y	
F8EE:85 2C	295	STA LMNEM	;FETCH 3-CHAR MNEMONIC
F8F0:B9 00 FA	296	LDA MNEMR,Y	; (PACKED INTO 2-BYTES)
F8F3:85 2D	297	STA RMNEM	
F8F5:A9 00	298	PRMN1 LDA #\$00	
F8F7:A0 05	299	LDY #\$05	
F8F9:06 2D	300	PRMN2 ASL RMNEM	;SHIFT 5 BITS OF CHARACTER INTO A
F8FB:26 2C	301	ROL LMNEM	
F8FD:2A	302	ROL A	; (CLEARS CARRY)
F8FE:88	303	DEY	
F8FF:D0 F8 F8F9	304	BNE PRMN2	
F901:69 BF	305	ADC #\$BF	;ADD "?" OFFSET
F903:20 ED FD	306	JSR COUT	;OUTPUT A CHAR OF MNEM
F906:CA	307	DEX	

F907:D0 EC	F8F5	308	BNE	PRMN1	
F909:20 48 F9		309	JSR	PRBLNK	;OUTPUT 3 BLANKS
F90C:A4 2F		310	LDY	LENGTH	
F90E:A2 06		311	LDX	#\$06	;CNT FOR 6 FORMAT BITS
F910:E0 03		312	PRADR1	CPX	#\$03
F912:F0 1C	F930	313	BEQ	PRADR5	;IF X=3 THEN ADDR.
F914:06 2E		314	PRADR2	ASL	FORMAT
F916:90 0E	F926	315	BCC	PRADR3	
F918:BD B3 F9		316	LDA	CHAR1-1,X	
F91B:20 ED FD		317	JSR	COUT	
F91E:BD B9 F9		318	LDA	CHAR2-1,X	
F921:F0 03	F926	319	BEQ	PRADR3	
F923:20 ED FD		320	JSR	COUT	
F926:CA		321	PRADR3	DEX	
F927:D0 E7	F910	322	BNE	PRADR1	
F929:60		323	RTS		
F92A:88		324	PRADR4	DEY	
F92B:30 E7	F914	325	BMI	PRADR2	
F92D:20 DA FD		326	JSR	PRBYTE	
F930:A5 2E		327	PRADR5	LDA	FORMAT
F932:C9 E8		328	CMP	#\$E8	;HANDLE REL ADR MODE
F934:B1 3A		329	LDA	(PCL),Y	;SPECIAL (PRINT TARGET,
F936:90 F2	F92A	330	BCC	PRADR4	; NOT OFFSET)
F938:20 56 F9		331	RELADR	JSR	PCADJ3
F93B:AA		332	TAX		;PCL,PCH+OFFSET+1 TO A,Y
F93C:E8		333	INX		
F93D:D0 01	F940	334	BNE	PRNTYX	;+1 TO Y,X
F93F:C8		335	INY		
F940:98		336	PRNTYX	TYA	
F941:20 DA FD		337	PRNTAX	JSR	PRBYTE
F944:8A		338	PRNTX	TXA	; OUTPUT TARGET ADR
F945:4C DA FD		339	JMP	PRBYTE	; OF BRANCH AND RETURN
F948:		340	*		
F948:A2 03		341	PRBLNK	LDX	#\$03
F94A:A9 A0		342	PRBL2	LDA	#\$A0
F94C:20 ED FD		343	PRBL3	JSR	COUT
F94F:CA		344	DEX		;OUTPUT A BLANK
F950:D0 F8	F94 A	345	BNE	PRBL2	;LOOP UNTIL COUNT=0
F952:60		346	RTS		
F953:		347	*		
F953:38		348	PCADJ	SEC	;0=1 BYTE, 1=2 BYTE,
F954:A5 2F		349	PCADJ2	LDA	LENGTH
F956:A4 3B		350	PCADJ3	LDY	; 2=3 BYTE
F958:AA		351	TAX		;TEST DISPLACEMENT SIGN
F959:10 01	F95 C	352	BPL	PCADJ4	; (FOR REL BRANCH)
F95B:88		353	DEY		;EXTEND NEG BY DECR PCH
F95C:65 3A		354	PCADJ4	ADC	PCL
F95E:90 01	F96 1	355	BCC	RTS2	;PCL+LENGTH(OR DISPL)+1 TO A
F960:C8		356	INY		; CARRY INTO Y (PCH)
F961:60		357	RTS2	RTS	
F962:		358			
F962:		359	; FMT1 BYTES: XXXXXY0 INSTRS		
F962:		360	; IF Y=0 THEN LEFT HALF BYTE		
F962:		361	; IF Y=1 THEN RIGHT HALF BYTE		

F962:	362 ;		(X=INDEX)
F962:	363 ;		
F962:04	364 FMT1	DFB	\$04
F963:20	365	DFB	\$20
F964:54	366	DFB	\$54
F965:30	367	DFB	\$30
F966:0D	368	DFB	\$0D
F967:80	369	DFB	\$80
F968:04	370	DFB	\$04
F969:90	371	DFB	\$90
F96A:03	372	DFB	\$03
F96B:22	373	DFB	\$22
F96C:54	374	DFB	\$54
F96D:33	375	DFB	\$33
F96E:0D	376	DFB	\$0D
F96F:80	377	DFB	\$80
F970:04	378	DFB	\$04
F971:90	379	DFB	\$90
F972:04	380	DFB	\$04
F973:20	381	DFB	\$20
F974:54	382	DFB	\$54
F975:33	383	DFB	\$33
F976:0D	384	DFB	\$0D
F977:80	385	DFB	\$80
F978:04	386	DFB	\$04
F979:90	387	DFB	\$90
F97A:04	388	DFB	\$04
F97B:20	389	DFB	\$20
F97C:54	390	DFB	\$54
F97D:3B	391	DFB	\$3B
F97E:0D	392	DFB	\$0D
F97F:80	393	DFB	\$80
F980:04	394	DFB	\$04
F981:90	395	DFB	\$90
F982:00	396	DFB	\$00
F983:22	397	DFB	\$22
F984:44	398	DFB	\$44
F985:33	399	DFB	\$33
F986:0D	400	DFB	\$0D
F987:C8	401	DFB	\$C8
F988:44	402	DFB	\$44
F989:00	403	DFB	\$00
F98A:11	404	DFB	\$11
F98B:22	405	DFB	\$22
F98C:44	406	DFB	\$44
F98D:33	407	DFB	\$33
F98E:0D	408	DFB	\$0D
F98F:C8	409	DFB	\$C8
F990:44	410	DFB	\$44
F991:A9	411	DFB	\$A9
F992:01	412	DFB	\$01
F993:22	413	DFB	\$22
F994:44	414	DFB	\$44
F995:33	415	DFB	\$33

F996:0D	416	DFB	\$0D	
F997:80	417	DFB	\$80	
F998:04	418	DFB	\$04	
F999:90	419	DFB	\$90	
F99A:01	420	DFB	\$01	
F99B:22	421	DFB	\$22	
F99C:44	422	DFB	\$44	
F99D:33	423	DFB	\$33	
F99E:0D	424	DFB	\$0D	
F99F:80	425	DFB	\$80	
F9A0:04	426	DFB	\$04	
F9A1:90	427	DFB	\$90	
F9A2:26	428	DFB	\$26	
F9A3:31	429	DFB	\$31	
F9A4:87	430	DFB	\$87	
F9A5:9A	431	DFB	\$9A	
F9A6:	432			
F9A6:	433			
F9A6:	434			
F9A6:00	435	FMT2	DFB	\$00 ;ERR
F9A7:21	436		DFB	\$21 ;IMM
F9A8:81	437		DFB	\$81 ;Z-PAGE
F9A9:82	438		DFB	\$82 ;ABS
F9AA:00	439		DFB	\$00 ;IMPLIED
F9AB:00	440		DFB	\$00 ;ACCUMULATOR
F9AC:59	441		DFB	\$59 ;(ZPAG,X)
F9AD:4D	442		DFB	\$4D ;(ZPAG),Y
F9AE:91	443		DFB	\$91 ;ZPAG,X
F9AF:92	444		DFB	\$92 ;ABS,X
F9B0:86	445		DFB	\$86 ;ABS,Y
F9B1:4A	446		DFB	\$4A ;(ABS)
F9B2:85	447		DFB	\$85 ;ZPAG,Y
F9B3:9D	448		DFB	\$9D ;RELATIVE
F9B4:AC	449	CHAR1	DFB	\$AC ;','
F9B5:A9	450		DFB	\$A9 ;')'
F9B6:AC	451		DFB	\$AC ;','
F9B7:A3	452		DFB	\$A3 ;'#'
F9B8:A8	453		DFB	\$A8 ;'('
F9B9:A4	454		DFB	\$A4 ;'\$'
F9BA:D9	455	CHAR2	DFB	\$D9 ;'Y'
F9BB:00	456		DFB	\$00
F9BC:D8	457		DFB	\$D8 ;'Y'
F9BD:A4	458		DFB	\$A4 ;'\$'
F9BE:A4	459		DFB	\$A4 ;'\$'
F9BF:00	460		DFB	\$00
F9C0:1C	461	MNEML	DFB	\$1C
F9C1:8A	462		DFB	\$8A
F9C2:1C	463		DFB	\$1C
F9C3:23	464		DFB	\$23
F9C4:5D	465		DFB	\$5D
F9C5:8B	466		DFB	\$8B
F9C6:1B	467		DFB	\$1B
F9C7:A1	468		DFB	\$A1
F9C8:9D	469		DFB	\$9D

F9C9:8A	470	DFB	\$8A	
F9CA:1D	471	DFB	\$1D	
F9CB:23	472	DFB	\$23	
F9CC:9D	473	DFB	\$9D	
F9CD:8B	474	DFB	\$8B	
F9CE:1D	475	DFB	\$1D	
F9CF:A1	476	DFB	\$A1	
F9D0:00	477	DFB	\$00	
F9D1:29	478	DFB	\$29	
F9D2:19	479	DFB	\$19	
F9D3:AE	480	DFB	\$AE	
F9D4:69	481	DFB	\$69	
F9D5:A8	482	DFB	\$A8	
F9D6:19	483	DFB	\$19	
F9D7:23	484	DFB	\$23	
F9D8:24	485	DFB	\$24	
F9D9:53	486	DFB	\$53	
F9DA:1B	487	DFB	\$1B	
F9DB:23	488	DFB	\$23	
F9DC:24	489	DFB	\$24	
F9DD:53	490	DFB	\$53	
F9DE:19	491	DFB	\$19	; (A) FORMAT ABOVE
F9DF:A1	492	DFB	\$A1	
F9E0:00	493	DFB	\$00	
F9E1:1A	494	DFB	\$1A	
F9E2:5B	495	DFB	\$5B	
F9E3:5B	496	DFB	\$5B	
F9E4:A5	497	DFB	\$A5	
F9E5:69	498	DFB	\$69	
F9E6:24	499	DFB	\$24	; (B) FORMAT
F9E7:24	500	DFB	\$24	
F9E8:AE	501	DFB	\$AE	
F9E9:AE	502	DFB	\$AE	
F9EA:A8	503	DFB	\$A8	
F9EB:AD	504	DFB	\$AD	
F9EC:29	505	DFB	\$29	
F9ED:00	506	DFB	\$00	
F9EE:7C	507	DFB	\$7C	; (C) FORMAT
F9EF:00	508	DFB	\$00	
F9F0:15	509	DFB	\$15	
F9F1:9C	510	DFB	\$9C	
F9F2:6D	511	DFB	\$6D	
F9F3:9C	512	DFB	\$9C	
F9F4:A5	513	DFB	\$A5	
F9F5:69	514	DFB	\$69	
F9F6:29	515	DFB	\$29	; (D) FORMAT
F9F7:53	516	DFB	\$53	
F9F8:84	517	DFB	\$84	
F9F9:13	518	DFB	\$13	
F9FA:34	519	DFB	\$34	
F9FB:11	520	DFB	\$11	
F9FC:A5	521	DFB	\$A5	
F9FD:69	522	DFB	\$69	
F9FE:23	523	DFB	\$23	; (E) FORMAT

F9FF:A0	524	DFB	\$A0	
FA00:D8	525	MNEMR	DFB	\$D8
FA01:62	526		DFB	\$62
FA02:5A	527		DFB	\$5A
FA03:48	528		DFB	\$48
FA04:26	529		DFB	\$26
FA05:62	530		DFB	\$62
FA06:94	531		DFB	\$94
FA07:88	532		DFB	\$88
FA08:54	533		DFB	\$54
FA09:44	534		DFB	\$44
FA0A:C8	535		DFB	\$C8
FA0B:54	536		DFB	\$54
FA0C:68	537		DFB	\$68
FA0D:44	538		DFB	\$44
FA0E:E8	539		DFB	\$E8
FA0F:94	540		DFB	\$94
FA10:00	541		DFB	\$00
FA11:B4	542		DFB	\$B4
FA12:08	543		DFB	\$08
FA13:84	544		DFB	\$84
FA14:74	545		DFB	\$74
FA15:B4	546		DFB	\$B4
FA16:28	547		DFB	\$28
FA17:6E	548		DFB	\$6E
FA18:74	549		DFB	\$74
FA19:F4	550		DFB	\$F4
FA1A:CC	551		DFB	\$CC
FA1B:4A	552		DFB	\$4A
FA1C:72	553		DFB	\$72
FA1D:F2	554		DFB	\$F2
FA1E:A4	555		DFB	\$A4
FA1F:8A	556		DFB	\$8A
FA20:00	557		DFB	\$00
FA21:AA	558		DFB	\$AA
FA22:A2	559		DFB	\$A2
FA23:A2	560		DFB	\$A2
FA24:74	561		DFB	\$74
FA25:74	562		DFB	\$74
FA26:74	563		DFB	\$74
FA27:72	564		DFB	\$72
FA28:44	565		DFB	\$44
FA29:68	566		DFB	\$68
FA2A:B2				
567	DFB	\$B2		
FA2B:32	568		DFB	\$32
FA2C:B2	569		DFB	\$B2
FA2D:00	570		DFB	\$00
FA2E:22	571		DFB	\$22
FA2F:00	572		DFB	\$00
FA30:1A	573		DFB	\$1A
FA31:1A	574		DFB	\$1A
FA32:26	575		DFB	\$26
FA33:26	576		DFB	\$26

; (A) FORMAT

; (B) FORMAT

; (C) FORMAT

FA34:72	577	DFB	\$72	
FA35:72	578	DFB	\$72	
FA36:88	579	DFB	\$88	; (D) FORMAT
FA37:C8	580	DFB	\$C8	
FA38:C4	581	DFB	\$C4	
FA39:CA	582	DFB	\$CA	
FA3A:26	583	DFB	\$26	
FA3B:48	584	DFB	\$48	
FA3C:44	585	DFB	\$44	
FA3D:44	586	DFB	\$44	
FA3E:A2	587	DFB	\$A2	; (E) FORMAT
FA3F:C8	588	DFB	\$C8	
FA40:	589 *			
FA40:	C3FA 590	NEWIRQ	EQU \$C3FA	;new IRQ entry
FA40:	591 *			
FA40:85 45	592	OLDIRQ	STA \$45	;(should never be used)
FA42:A5 45	593	LDA	\$45	;for those who save A to \$45
FA44:4C FA C3	594	JMP	NEWIRQ	;go to interrupt handler
FA47:	595 *			
FA47:8D 06 C0	596	NEWBREAK	STA SETSLOT CXROM	;force in slots
FA4A:85 45	597		STA ACC	;save accumulator
FA4C:	598 *			
FA4C:28	599	BREAK	PLP	
FA4D:20 4C FF	600	JSR	SAV1	;SAVE REG'S ON BREAK
FA50:68	601	PLA		; INCLUDING PC
FA51:85 3A	602	STA	PCL	
FA53:68	603	PLA		
FA54:85 3B	604	STA	PCH	
FA56:6C F0 03	605	JMP	(BRKV)	;BRKV WRITTEN OVER BY DISK BOOT
FA59:	606 *			
FA59:20 82 F8	607	OLDBRK	JSR INSDS1	;PRINT USER PC
FA5C:20 DA FA	608	JSR	RGDSP1	; AND REGS
FA5F:4C 65 FF	609	JMP	MON	;GO TO MONITOR (NO PASS GO, NO \$200!)
FA62:D8	610	RESET	CLD	;DO THIS FIRST THIS TIME
FA63:20 84 FE	611	JSR	SETNORM	
FA66:20 2F FB	612	JSR	INIT	
FA69:20 93 FE	613	JSR	SETVID	
FA6C:20 89 FE	614	JSR	SETKBD	
FA6F:AD 58 C0	615	INITAN	LDA SETANO	; ANO = TTL LO
FA72:AD 5A C0	616		LDA SETANI	; ANI = TTL LO
FA75:A0 09	617		LDY #9	;CODE=INIT/RAA0981
FA77:20 B4 FB	618	JSR	GOTOCX	;DO APPLE2E INIT/RAA0981
FA7A:EA	619		NOP	; /RAA0981
FA7B:AD FF CF	620	LDA	CLRROM	; TURN OFF EXTNSN ROM
FA7E:2C 10 C0	621	BIT	KBDSTRB	; CLEAR KEYBOARD
FA81:D8	622	NEWMON	CLD	
FA82:20 3A FF	623	JSR	BELL	; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03	624	LDA	SOFTEV+1	;IS RESET HI
FA88:49 A5	625	EOR	#\$A5	;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03	626	CMP	PWREDUP	; PWR UP BYTE ???
FA8D:D0 17 FAA6	627	BNE	PWRUP	; NO SO PWRUP
FA8F:AD F2 03	628	LDA	SOFTEV	; YES SEE IF COLD START
FA92:D0 0F FAA3	629	BNE	NOFIX	; HAS BEEN DONE YET?
FA94:A9 E0	630	LDA	#\$E0	; DOES SOFT ENTRY VECTOR POINT AT BASIC?

```

FA96:CD F3 03      631      CMP  SORTEV+1
FA99:D0 08      FAA3 632      BNE  NOFIX      ; YES SO REENTER SYSTEM
FA9B:A0 03      633  FIXSEV LDY  #3      ; NO SO POINT AT WARM START
FA9D:8C F2 03      634      STY  SORTEV      ; FOR NEXT RESET
FAA0:4C 00 E0      635      JMP  BASIC      ; AND DO THE COLD START
FAA3:6C F2 03      636  NOFIX JMP  (SORTEV)    ; SOFT ENTRY VECTOR
FAA6:      637  *****
FAA6:20 60 FB      638  PWRUP JSR  APPLEII
FAA9:      FAA9 639  SETPG3 EQU  *      ; SET PAGE 3 VECTORS
FAA9:A2 05      640      LDX  #5
FAAB:BD FC FA      641  SETPLP LDA  PWRCON-1,X ; WITH CNTRL B ADRS
FAAE:9D EF 03      642      STA  BRKV-1,X ; OF CURRENT BASIC
FAB1:CA      643      DEX
FAB2:D0 F7      FAAB 644      BNE  SETPLP
FAB4:A9 C8      645      LDA  #5C8      ; LOAD HI SLOT +1
FAB6:86 00      646      STX  LOC0      ; SETPG3 MUST RETURN X=0
FAB8:85 01      647      STA  LOC1      ; SET PTR H
FABA:      648  *
FABA:      649  * Check 3 ID bytes instead of 4. Allows devices
FABA:      650  * other than Disk II's to be bootable.
FABA:      651  *
FABA:A0 05      652  SLOOP LDY  #5      ;Y is byte ptr
FABC:C6 01      653      DEC  LOC1
FABE:A5 01      654      LDA  LOC1
FAC0:C9 C0      655      CMP  #5C0      ; AT LAST SLOT YET?
FAC2:F0 D7      FA9B 656      BEQ  FIXSEV      ; YES AND IT CAN'T BE A DISK
FAC4:8D F8 07      657      STA  MSLOT
FAC7:B1 00      658  NXTBYT LDA  (LOC0),Y ; FETCH A SLOT BYTE
FAC9:D9 01 FB      659      CMP  DISKID-1,Y ; IS IT A DISK ??
FACC:D0 EC      FAB4 660      BNE  SLOOP      ; NO, SO NEXT SLOT DOWN
FACE:88      661      DEY
FACF:88      662      DEY      ; YES, SO CHECK NEXT BYTE
FAD0:10 F5      FAC7 663      BPL  NXTBYT      ; UNTIL 3 BYTES CHECKED
FAD2:6C 00 00      664      JMP  (LOC0)      ; GO BOOT...
FAD5:      665  *
FAD5:EA      666      NOP
FAD6:EA      667      NOP
FAD7:      668  *
FAD7:20 8E FD      669  REGDSP JSR  CROUT      ;DISPLAY USER REG CONTENTS
FADA:A9 45      670  RGDSP1 LDA  #545      ;WITH LABELS
FADC:85 40      671      STA  A3L
FADE:A9 00      672      LDA  #500
FAE0:85 41      673      STA  A3H
FAE2:A2 FB      674      LDX  #5FB
FAE4:A9 A0      675  RDSP1 LDA  #5A0
FAE6:20 ED FD      676      JSR  COUT
FAE9:BD 1E FA      677      LDA  RTBL-251,X
FAEC:20 ED FD      678      JSR  COUT
FAEF:A9 BD      679      LDA  #5BD
FAF1:20 ED FD      680      JSR  COUT
FAF4:B5 4A      681      LDA  ACC+5,X
FAF6:20 DA FD      682      JSR  PRBYTE
FAF9:E8      683      INX
FAFA:30 E8      FAE4 684      BMI  RDSP1

```

FAFC:60	685	RTS	
FAFD:	686 *		
FAFD:59 FA	687	PWRCON	DW OLDBRK
FAFF:00 E0 45	688	DFB	\$00,\$E0,\$45
FB02:20 FF 00 FF	689	DISKID	DFB \$20,\$FF,\$00,\$FF
FB06:03 FF 3C	690	DFB	\$03,\$FF,\$3C
FB09:C1 F0 F0 EC	691	ASC	'Apple']['
FB11: FB11	692	XLTBL	EQU *
FB11:C4 C2 C1	693	DFB	\$C4,\$C2,\$C1
FB14:FF C3	694	DFB	\$FF,\$C3
FB16:FF FF FF	695	DFB	\$FF,\$FF,\$FF
FB19:	696 *		
FB19:C1 D8 D9	697	RTBL	DFB \$C1,\$D8,\$D9 ;REGISTER NAMES FOR REGDSP:
FB1C:D0 D3	698	DFB	\$D0,\$D3 ;'AXYPS'
FB1E:AD 70 C0	699	PREAD	LDA PTRIG ;TRIGGER PADDLES
FB21:A0 00	700	LDY	#\$00 ;INIT COUNT
FB23:EA	701	NOP	;COMPENSATE FOR 1ST COUNT
FB24:EA	702	NOP	
FB25:BD 64 C0	703	PREAD2	LDA PADDLO,X ;COUNT Y-REG EVERY 12 USEC.
FB28:10 04 FB2E	704	BPL	RTS2D
FB2A:C8	705	INY	
FB2B:D0 F8 FB25	706	BNE	PREAD2 ;EXIT AT 255 MAX
FB2D:88	707	DEY	
FB2E:60	708	RTS2D	RTS
FB2F:	1 *		
FB2F:A9 00	2	INIT	LDA #\$00 ;CLR STATUS FOR DEBUG SOFTWARE
FB31:85 48	3	STA	STATUS
FB33:AD 56 C0	4	LDA	LORES
FB36:AD 54 C0	5	LDA	LOWSCR ;INIT VIDEO MODE
FB39:AD 51 C0	6	SETTXT	LDA TXTSET ;SET FOR TEXT MODE
FB3C:A9 00	7	LDA	#\$00 ;FULL SCREEN WINDOW
FB3E:F0 08 FB4B	8	BEQ	SETWND
FB40:AD 50 C0	9	SETGR	LDA TXTCLR ;SET FOR GRAPHICS MODE
FB43:AD 53 C0	10	LDA	MIXSET ;LOWER 4 LINES AS TEXT WINDOW
FB46:20 36 F8	11	JSR	CLRTOP
FB49:A9 14	12	LDA	#\$14
FB4B:85 22	13	SETWND	STA WNDTOP ;SET FOR 40 COL WINDOW
FB4D:A9 00	14	LDA	#\$00 ;TOP IN A-REG,
FB4F:85 20	15	STA	WNDLFT ; BOTTOM AT LINE \$24
FB51:A0 0C	16	LDY	#\$C ;CODE=SETWND /RRA0981
FB53:D0 5F FBB4	17	BNE	GOTOCX
FB55:A9 18	18	LDA	#\$18
FB57:85 23	19	STA	WNCBTM
FB59:A9 17	20	LDA	#\$17 ;VTAB TO ROW 23
FB5B:85 25	21	TABV	STA CV ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC	22	JMP	VTAB
FB60:	23 *		
FB60:20 58 FC	24	APPLEII	JSR HOME ;CLEAR THE SCRIN
FB63:A0 09	25	LDY	#9
FB65:B9 09 FF	26	STITLE	LDA TITLE-1,Y ;GET A CHAR
FB68:99 0E 04	27	STA	LINE1+14,Y ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88	28	DEY	
FB6C:D0 F7 FB65	29	BNE	STITLE
FB6E:60	30	RTS	

```

FB6F:          31 *
FB6F:AD F3 03  32 SETPWRC LDA  SOTEV+1      ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5      33          EOR  #$A5      ;COMPLEMENT' FOR THE RESET VECTOR
FB74:8D F4 03  34          STA  PWREDUP
FB77:60         35          RTS
FB78:          36 *
FB78:          FB7 8 37 VIDWAIT EQU  *      ;CHECK FOR A PAUSE (CONTROL-S).
FB78:C9 8D      38          CMP  #$8D      ;ONLY WHEN I HAVE A CR
FB7A:D0 18  FB9 4 39          BNE  NOWAIT      ;NOT SO, DO REGULAR
FB7C:AC 00 C0    40          LDY  KBD      ;IS KEY PRESSED?
FB7F:10 13  FB9 4 41          BPL  NOWAIT      ;NO.
FB81:C0 93      42          CPY  #$93      ;YES -- IS IT CTRL-S?
FB83:D0 0F  FB9 4 43          BNE  NOWAIT      ;NOPE - IGNORE
FB85:2C 10 C0    44          BIT  KBDSTRB    ;CLEAR STROBE
FB88:AC 00 C0    45 KBDWAIT LDY  KBD      ;WAIT TILL NEXT KEY TO RESUME
FB8B:10 FB  FB8 8 46          BPL  KBDWAIT    ;WAIT FOR KEYPRESS
FB8D:C0 83      47          CPY  #$83      ;IS IT CONTROL-C?
FB8F:F0 03  FB9 4 48          BEQ  NOWAIT      ;YES, SO LEAVE IT
FB91:2C 10 C0    49          BIT  KBDSTRB    ;CLR STROBE
FB94:4C FD FB    50 NOWAIT JMP  VIDOUT      ;DO AS BEFORE
FB97:          51 *
FB97:38         52 ESCOLD SEC          ;INSURE CARRY SET
FB98:4C 2C FC    53          JMP  ESC1
FB9B:A8         54 ESCNOW TAY          ;USE CHAR AS INDEX
FB9C:B9 48 FA    55          LDA  XLTLB-$C9,Y  ;TRANSLATE IJKM TO CBAD
FB9F:20 97 FB    56          JSR  ESCOLD    ;DO THE CURSOR MOTION
FBA2:20 21 FD    57          JSR  RDESC    ;GET IJKM, i,jkm, ARROWS/RAA0981
FBA5:C9 CE       58 ESCNEW CMP  #$CE      ;IS THIS AN 'N'?
FBA7:B0 EE  FB9 7 59          BCS  ESCOLD    ;'N' OR GREATER - DO IT!
FBA9:C9 C9       60          CMP  #$C9      ;LESS THAN 'I'?
FBAB:90 EA  FB9 7 61          BCC  ESCOLD    ;YES, SO DO OLD WAY
FBAD:C9 CC       62          CMP  #$CC      ;IS IT AN 'L'?
FBAF:F0 E6  FB9 7 63          BEQ  ESCOLD    ;DO NORMAL
FBB1:D0 E8  FB9 B 64          BNE  ESCNOW    ;GO DO IT
FBB3:          65 *
FBB3:          C00 6 66 SETSL0TCXROM EQU $C006  ;/RAA0981
FBB3:          C00 7 67 SETINTCXROM EQU $C007  ;/RAA0981
FBB3:          C01 5 68 RDCXROM EQU  $C015    ;/RAA0981
FBB3:          69 *          /RAA0981
FBB3:06         70 VERSION DFB  $06      ;FOR IDCHECK/RAA0981
FBB4:          71 *
FBB4:          FBB 4 72 GOTOCX EQU  *      ;/RAA0981
FBB4:2C 15 C0    73          BIT  RDCXROM    ;GET CURRENT STATE/RAA0981
FBB7:08         74          PHP          ;SAVE ROMBANK STATE/RAA0981
FBB8:8D 07 C0    75          STA  SETINTCXROM ;SET ROMS ON/RAA0981
FBBB:4C 00 C1    76          JMP  C1ORG    ;=>OFF TO CXSPACE/RAA0981
FBBE:          77 *
FBBE:00         78          DFB  0
FBBF:00         79          DFB  0
FBC0:          80 *
FBC0:E0         81 ZIDBYTE DFB  $E0      ;/e ROM rev ID byte
FBC1:          82 *
FBC1:48         83 BASCALC PHA          ;CALC BASE ADDR IN BASL,H
FBC2:4A         84          LSR  A      ;FOR GIVEN LINE NO.

```

FBC3:29 03	85	AND	#\$03	; 0<=LINE NO.<=\$17
FBC5:09 04	86	ORA	#\$04	;ARG = 000ABCDE, GENERATE
FBC7:85 29	87	STA	BASH	; BASH = 000001CD
FBC9:68	88	PLA		; AND
FBCA:29 18	89	AND	#\$18	; BASL = EABAB000
FBCC:90 02	FBD0 90	BCC	BASCLC2	
FBCE:69 7F	91	ADC	#\$7F	
FBD0:85 28	92 BASCLC2	STA	BASL	
FBD2:0A	93	ASL	A	
FBD3:0A	94	ASL	A	
FBD4:05 28	95	ORA	BASL	
FBD6:85 28	96	STA	BASL	
FBD8:60	97	RTS		
FBD9:	98 *			
FBD9:C9 87	99 BELL1	CMP	#\$87	;BELL CHAR? (CONTROL-G)
FBDB:D0 12	FBEF 100	BNE	RTS2B	; NO, RETURN.
FBDD:A9 40	101	LDA	#\$40	; YES...
FBDf:20 A8 FC	102	JSR	WAIT	;DELAY .01 SECONDS
FBE2:A0 C0	103	LDY	#\$C0	
FBE4:A9 0C	104 BELL2	LDA	#\$0C	;TOGGLE SPEAKER AT 1 KHZ
FBE6:20 A8 FC	105	JSR	WAIT	; FOR .1 SEC.
FBE9:AD 30 C0	106	LDA	SPKR	
FBEC:88	107	DEY		
FBED:D0 F5	FBE4 108	BNE	BELL2	
FBEF:60	109	RTS2B	RTS	
FBF0:	110 *			
FBF0:A4 24	111 STORADV	LDY	CH	;CURSOR H INDEX TO Y-REG
FBF2:91 28	112	STA	(BASL),Y	;STORE CHAR IN LINE
FBF4:E6 24	113 ADVANCE	INC	CH	;INCREMENT CURSOR H INDEX
FBF6:A5 24	114	LDA	CH	; (MOVE RIGHT)
FBF8:C5 21	115	CMP	WNDWDTH	;BEYOND WINDOW WIDTH?
FBFA:B0 66	FC62 116	BCS	CR	; YES, CR TO NEXT LINE.
FBFC:60	117	RTS3	RTS	; NO, RETURN.
FBFD:	118 *			
FBFD:C9 A0	119 VIDOUT	CMP	#\$A0	;CONTROL CHAR?
FBFF:B0 EF	FBF0 120	BCS	STORADV	; NO, OUTPUT IT.
FC01:A8	121	TAY		;INVERSE VIDEO?
FC02:10 EC	FBF0 122	BPL	STORADV	; YES, OUTPUT IT.
FC04:C9 8D	123	CMP	#\$8D	;CR?
FC06:F0 5A	FC62 124	BEQ	CR	; YES.
FC08:C9 8A	125	CMP	#\$8A	;LINE FEED?
FC0A:F0 5A	FC66 126	BEQ	LF	; IF SO, DO IT.
FC0C:C9 88	127	CMP	#\$88	;BACK SPACE? (CONTROL-H)
FC0E:D0 C9	FBD9 128	BNE	BELL1	; NO, CHECK FOR BELL.
FC10:C6 24	129 BS	DEC	CH	;DECREMENT CURSOR H INDEX
FC12:10 E8	FBFC 130	BPL	RTS3	;IF POSITIVE, OK; ELSE MOVE UP.
FC14:A5 21	131	LDA	WNDWDTH	;SET CH TO WINDOW WIDTH - 1.
FC16:85 24	132	STA	CH	
FC18:C6 24	133	DEC	CH	; (RIGHTMOST SCREEN POS)
FC1A:A5 22	134 UP	LDA	WNDTOP	;CURSOR V INDEX
FC1C:C5 25	135	CMP	CV	
FC1E:B0 DC	FBFC 136	BCS	RTS3	;IF TOP LINE THEN RETURN
FC20:C6 25	137	DEC	CV	;DECR CURSOR V INDEX
FC22:	138 *			

FC22:A5 25		139 VTAB	LDA CV	;GET CURSOR V INDEX
FC24:85 28		140 VTABZ	STA BASL	;temporarily save Acc
FC26:98		141	TYA	;and Y
FC27:A0 04		142	LDY #\$4	;this is VTABZ call
FC29:D0 89	FBB4	143 GOTOCX1	BNE GOTOCX	;=> always perform call
FC2B:		144 *		
FC2B:EA		145	NOP	
FC2C:		146 *		
FC2C:49 C0		147 ESC1	EOR #\$C0	;ESC '@'?
FC2E:F0 28	FC58	148	BEQ HOME	; IF SO DO HOME AND CLEAR
FC30:69 FD		149	ADC #\$FD	;ESC-A OR B CHECK
FC32:90 C0	FBF4	150	BCC ADVANCE	; A, ADVANCE
FC34:F0 DA	FC10	151	BEQ BS	; B, BACKSPACE
FC36:69 FD		152	ADC #\$FD	;ESC-C OR D CHECK
FC38:90 2C	FC66	153	BCC LF	; C, DOWN
FC3A:F0 DE	FC1A	154	BEQ UP	; D, GO UP
FC3C:69 FD		155	ADC #\$FD	;ESC-E OR F CKECK
FC3E:90 5C	FC9C	156	BCC CLREOL	; E, CLEAR TO END OF LINE
FC40:D0 BA	FBFC	157	BNE RTS3	; ELSE NOT F,RETURN
FC42:		158 *		
FC42:	FC42	159 CLREOP	EQU *	;/RRA0981
FC42:A0 0A		160	LDY #\$A	;CODE=CLREOP/RRA0981
FC44:D0 E3	FC29	161	BNE GOTOCX1	;DO 40/80 /RRA0981
FC46:		162 *		
FC46:2C 1F	C0	163 NEWVW	BIT RD80VID	;in 80 columns?
FC49:10 04	FC4F	164	BPL NEWVW1	;=>not 80 columns
FC4B:A0 00		165	LDY #\$0	;Print a character
FC4D:F0 0B	FC5A	166	BEQ GOTOCX3	;through video firmware
FC4F:98		167 NEWVW1	TYA	;get masked character
FC50:48		168	PHA	;and set up for vidwait
FC51:20 78	FB	169	JSR VIDWAIT	;print the character
FC54:68		170	PLA	;restore Acc
FC55:A4 35		171	LDY YSAV1	;and Y
FC57:60		172	RTS	
FC58:		173 *		
FC58:	FC58	174 HOME	EQU *	;/RRA0981
FC58:A0 05		175	LDY #5	;CODE=HOME/RRA0981
FC5A:4C B4	FB	176 GOTOCX3	JMP GOTOCX	;do 40/80
FC5D:		177 *		
FC5D:EA		178	NOP	
FC5E:EA		179	NOP	
FC5F:EA		180	NOP	
FC60:EA		181	NOP	
FC61:EA		182	NOP	
FC62:		183 *		
FC62:A9 00		184 CR	LDA #\$00	;CURSOR TO LEFT OF INDEX
FC64:85 24		185	STA CH	;(RET CURSOR H=0)
FC66:E6 25		186 LF	INC CV	;INCR CURSOR V. (DOWN 1 LINE)
FC68:A5 25		187	LDA CV	
FC6A:C5 23		188	CMP WNDBTM	;OFF SCREEN?
FC6C:90 B6	FC24	189	BCC VTABZ	; NO, SET BASE ADDR
FC6E:C6 25		190	DEC CV	;DECR CURSOR V. (BACK TO BOTTOM)
FC70:		191 *		
FC70:	FC 70	192 SCROLL	EQU *	;/RRA0981

```

FC70:A0 06      193      LDY #6      ;CODE=SCROLL/RRA0981
FC72:D0 B5      FC29    194      BNE GOTOCX1 ;DO 40/80 /RRA0981
FC74:           195 *
FC74:           196 * Jump here to swap out ROMs
FC74:           197 * for interrupt handlers in peripheral cards
FC74:           198 *
FC74:8D 06 C0    199      IRQUUSER STA SETSLOT CXROM ;switch in slots
FC77:6C FE 03    200      JMP ($3FE) ;and jump to user
FC7A:           201 *
FC7A:           202 * IRQDONE ($C3F4) jumps here after interrupt
FC7A:           203 * because this cannot be done from $Cn00 space
FC7A:           204 *
FC7A:68          205      IRQDONE2 PLA ;Fix $C800 space
FC7B:8D F8 07    206      STA MSL0T ;restore MSL0T
FC7E:C9 C1      207      CMP #$C1 ;valid Cn?
FC80:90 0D      FC8F    208      BCC IRQNOSLT
FC82:8D FF CF    209      STA $CFFF ;Deselect all $C800
FC85:A0 00      210      LDY #0
FC87:A6 01      211      LDX $1
FC89:85 01      212      STA $1
FC8B:B1 00      213      LDA ($0),Y ;do $Cn00 reference
FC8D:86 01      214      STX $1 ;fix zp location
FC8F:8D 07 C0    215      IRQNOSLT STA SETINT CXROM
FC92:4C 7C C4    216      JMP IRQFIX ;and restore the machine state
FC95:           217 *
FC95:90 02      FC99    218      DOCOUT1 BCC DOCOUT2 ;don't mask controls
FC97:25 32      219      AND INVFLG ;apply inverse mask
FC99:4C F7 FD    220      DOCOUT2 JMP COUTZ1 ;go back to COUT1
FC9C:           221 *
FC9C:           0000    222      DS F80RG+$49C-*,0 ;pad to clreol
FC9C:           223 *
FC9C:           224 * Note: bytes CLREOL and CLREOLZ ($38 and $18)
FC9C:           225 * are used by slot test at $FBB7.
FC9C:           226 *
FC9C:38          227      CLREOL SEC ;say it is EOL
FC9D:90          228      DFB $90 ;'BCC' opcode
FC9E:18          229      CLREOLZ CLC ;say it is EOLZ
FC9F:84 2A      230      STY BAS2L ;save Y in temp
FCA1:A0 07      231      LDY #7 ;code=CLREOL
FCA3:B0 78      FD1D    232      BCS GOTOCX2 ;do it
FCA5:C8          233      INY ;code 8=CLREOLZ
FCA6:D0 75      FD1D    234      BNE GOTOCX2
FCA8:           235 *
FCA8:38          236      WAIT SEC ;enter with count in A
FCA9:48          237      WAIT2 PHA ;delay is:
FCAA:E9 01      238      WAIT3 SBC #$01
FCAC:D0 FC      FCAA    239      BNE WAIT3 ;13+11*A+5*A*A cycles
FCAE:68          240      PLA ;@ 1.023 usec per cycle
FCAF:E9 01      241      SBC #$01
FCB1:D0 F6      FCA9    242      BNE WAIT2
FCB3:60          243      RTS
FCB4:           244 *
FCB4:E6 42      245      NXTA4 INC A4L ;INCR 2-BYTE A4
FCB6:D0 02      FCBA    246      BNE NXTA1 ; AND A1

```


FCEB8:E6 43	247	INC	A4H	
FCBA:A5 3C	248	NXTA1 LDA	A1L	;INCR 2-BYTE A1.
FCBC:C5 3E	249	CMP	A2L	; AND COMPARE TO A2
FCBE:A5 3D	250	LDA	A1H	; (CARRY SET IF >=)
FCC0:E5 3F	251	SBC	A2H	
FCC2:E6 3C	252	INC	A1L	
FCC4:D0 02 FCC8	253	BNE	RTS4B	
FCC6:E6 3D	254	INC	A1H	
FCC8:60	255	RTS4B	RTS	
FCC9:	256	*		
FCC9:8D 07 C0	257	HEADR STA	SETINTCXROM	;force internal ROM
FCCC:20 67 C5	258	JSR	XHEADER	;write header
FCCF:4C C5 FE	259	JMP	RET CX1	;force slots and return
FCD2:	260	*		
FCD2:	261	*		* For the disassembler to be able to do I/O to slots,
FCD2:	262	*		* it cannot make calls to the I/O routines with the
FCD2:	263	*		* internal ROM switched in. This stuff switches the
FCD2:	264	*		* ROM out for such instances.
FCD2:	265	*		
FCD2:8D 06 C0	266	ERR3 STA	SETSL0TCXROM	;force slot ROM
FCD5:20 4A F9	267	JSR	PRBL2	;tab to the error
FCD8:A9 DE	268	LDA	#\$DE	;to print a caret "^"
FCDA:20 ED FD	269	JSR	COUT	;print it
FCDD:20 3A FF	270	JSR	BELL	;and beep
FCE0:4C F0 FC	271	JMP	GETINST1	;and go get next instruction
FCE3:	272	*		
FCE3:8D 06 C0	273	DISLIN STA	SETSL0TCXROM	;force slot ROM
FCE6:20 D0 F8	274	JSR	INSTDSP	;disassemble the instruction
FCE9:20 53 F9	275	JSR	PCADJ	;calculate new PC
FCEC:84 3B	276	STY	PCH	;and update PC
FCEE:85 3A	277	STA	PCL	
FCF0:	278	*		
FCF0:	279	*		* NOTE: The entry point GETINST1 is hard-coded in
FCF0:	280	*		* BFUNC of the Video firmware.
FCF0:	281	*		
FCF0:A9 A1	282	GETINST1 LDA	#\$A1	;get mini-prompt "!"
FCF2:85 33	283	STA	PROMPT	
FCF4:20 67 FD	284	JSR	GETLNZ	;go get a line of input
FCF7:8D 07 C0	285	STA	SETINTCXROM	;force internal ROM
FCFA:4C 9C CF	286	JMP	DOINST	;and return to CX space
FCFD:	287	*		
FCFD:B9 00 02	288	UPMON LDA	IN,Y	;get character
FD00:C8	289	INY		;point to next char
FD01:C9 E1	290	CMP	#\$E1	;is it lowercase?
FD03:90 06 FDOB	291	BCC	UPMON2	;=>nope
FD05:C9 FB	292	CMP	#\$FB	;lowercase?
FD07:B0 02 FDOB	293	BCS	UPMON2	;=>nope
FD09:29 DF	294	AND	#\$DF	;else upshift
FD0B:60	295	UPMON2	RTS	
FD0C:	296	*		
FD0C:A0 0B	297	RDKEY LDY	#\$B	;code=RDKEY
FD0E:D0 03 FD13	298	BNE	RDKEY0	;allow \$FD10 entry
FD10:4C 18 FD	299	FD10 JMP	RDKEY1	;if enter here, do nothing
FD13:20 B4 FB	300	RDKEY0 JSR	GOTOCX	;display cursor

FD16:EA	301	NOP	
FD17:EA	302	NOP	
FD18:6C 38 00	303	RDKEY1 JMP	(KSWL) ;GO TO USER KEY-IN
FD1B:	304	*	
FD1B: FD1B	305	KEYIN EQU	*
FD1B:A0 03	306	LDY	#3 ;RDKEY/RAA0981
FD1D:4C B4 FB	307	GOTOCX2 JMP	GOTOCX ;/RAA0981
FD20:EA	308	NOP	;/RAA0981
FD21:	309	*	
FD21: FD21	310	RDESC EQU	*
FD21:20 0C FD	311	JSR RDKEY	;GET A KEY
FD24:A0 01	312	LDY	#1 ;CODE=FIXIT
FD26:D0 F5 FD1D	313	BNE GOTOCX2	;=>always
FD28:	314	*	
FD28:	315	* Flag to the video firmware that escapes are allowed.	
FD28:	316	* This routine is called by RDCHAR which is called by	
FD28:	317	* GETLN. The high bit of MSL0T is set by all cards	
FD28:	318	* that use the C800 space.	
FD28:	319	*	
FD28:4E F8 07	320	NEWRDKEY LSR	MSL0T ;<128 means escape allowed
FD2B:4C 0C FD	321	JMP RDKEY	;now read the key
FD2E:EA	322	NOP	
FD2F:	323	*	
FD2F:20 21 FD	324	ESC JSR	RDESC ;/RAA0981
FD32:20 A5 FB	325	JSR	ESCNEW ;HANDLE ESC FUNCTION.
FD35:20 28 FD	326	RDCHAR JSR	NEWRDKEY ;Flag RDCHAR and read key
FD38:C9 9B	327	CMP	#\$9B ;'ESC'?
FD3A:F0 F3 FD2F	328	BEQ	ESC ; YES, DON'T RETURN.
FD3C:60	329	RTS	
FD3D:	330	*	
FD3D:A0 0F	331	PICKFIX LDY	#\$F ;code = fixpick
FD3F:20 B4 FB	332	JSR	GOTOCX ;do 80 column pick
FD42:A4 24	333	LDY	CH ;restore Y
FD44:9D 00 02	334	STA	IN,X ;and save new character
FD47:	335	*#03 AUTOST2 Auto-Start Monitor ROM 27-AUG-84	
PAGE 20			
FD47:20 ED FD	336	NOTCR JSR	COUT ;echo typed char
FD4A:EA	337	NOP	
FD4B:EA	338	NOP	
FD4C:EA	339	NOP	
FD4D:BD 00 02	340	LDA	IN,X
FD50:C9 88	341	CMP	#\$88 ;CHECK FOR EDIT KEYS
FD52:F0 1D FD71	342	BEQ	BCKSPC ; - BACKSPACE
FD54:C9 98	343	CMP	#\$98
FD56:F0 0A FD62	344	BEQ	CANCEL ; - CONTROL-X
FD58:E0 F8	345	CPX	#\$F8
FD5A:90 03 FD5F	346	BCC	NOTCRI ;MARGIN?
FD5C:20 3A FF	347	JSR	BELL ; YES, SOUND BELL
FD5F:E8	348	NOTCRI INX	;ADVANCE INPUT INDEX
FD60:D0 13 FD75	349	BNE	NXTCHAR
FD62:	350	*	
FD62:A9 DC	351	CANCEL LDA	#\$DC ;BACKSLASH AFTER CANCELLED LINE
FD64:20 ED FD	352	JSR	COUT
FD67:20 8E FD	353	GETLNZ JSR	CROUT ;OUTPUT 'CR'

FD6A:A5 33	354	GETLN	LDA	PROMPT	;OUTPUT PROMPT CHAR
FD6C:20 ED FD	355		JSR	COUT	
FD6F:A2 01	356		LDX	#\$01	;INIT INPUT INDEX
FD71:8A	357	BCKSPC	TXA		
FD72:F0 F3 FD67	358		BEQ	GETLNZ	;WILL BACKSPACE TO 0
FD74:CA	359		DEX		
FD75:20 35 FD	360	NXTCHAR	JSR	RDCHAR	
FD78:C9 95	361		CMP	#\$95	;USE SCREEN CHAR
FD7A:D0 08 FD84	362		BNE	ADDINP	; FOR CONTROL-U
FD7C:B1 28	363		LDA	(BASL),Y	;do 40 column pick
FD7E:2C 1F C0	364		BIT	RD80VID	;80 columns?
FD81:30 BA FD3D	365		BMI	PICKFIX	;=>yes, fix it
FD83:EA	366		NOP		
FD84:9D 00 02	367	ADDINP	STA	IN,X	;ADD TO INPUT BUFFER
FD87:C9 8D	368		CMP	#\$8D	
FD89:D0 BC FD47	369		BNE	NOTCR	
FD8B:20 9C FC	370		JSR	CLREOL	;CLR TO EOL IF CR
FD8E:A9 8D	371	CROUT	LDA	#\$8D	
FD90:D0 5B FDED	372		BNE	COUT	; (ALWAYS)
FD92:	373	*			
FD92:A4 3D	374	PRA1	LDY	A1H	;PRINT CR,A1 IN HEX
FD94:A6 3C	375		LDX	A1L	
FD96:20 8E FD	376	PRYX2	JSR	CROUT	
FD99:20 40 F9	377		JSR	PRNTYX	
FD9C:A0 00	378		LDY	#\$00	
FD9E:A9 AD	379		LDA	#\$AD	;PRINT '-'
FDA0:4C ED FD	380		JMP	COUT	
FDA3:	381	*			
FDA3:A5 3C	382	XAM8	LDA	A1L	
FDA5:09 07	383		ORA	#\$07	;SET TO FINISH AT
FDA7:85 3E	384		STA	A2L	; MOD 8=7
FDA9:A5 3D	385		LDA	A1H	
FDAB:85 3F	386		STA	A2H	
FDAD:A5 3C	387	MO			
D8CHK LDA A1L					
FDAF:29 07	388		AND	#\$07	
FDB1:D0 03 FDB6	389		BNE	DATAOUT	
FDB3:20 92 FD	390	XAM	JSR	PRA1	
FDB6:A9 A0	391	DATAOUT	LDA	#\$A0	
FDB8:20 ED FD	392		JSR	COUT	;OUTPUT BLANK
FDBB:B1 3C	393		LDA	(A1L),Y	
FDBD:20 DA FD	394		JSR	PRBYTE	;OUTPUT BYTE IN HEX
FDC0:20 BA FC	395		JSR	NXTA1	
FDC3:90 E8 FDAD	396		BCC	MOD8CHK	;NOT DONE YET. GO CHECK MOD 8
FDC5:60	397	RTS4C	RTS		;DONE.
FDC6:	398	*			
FDC6:4A	399	XAMPM	LSR	A	;DETERMINE IF MONITOR MODE IS
FDC7:90 EA FDB3	400		BCC	XAM	; EXAMINE, ADD OR SUBTRACT
FDC9:4A	401		LSR	A	
FDCA:4A	402		LSR	A	
FDCB:A5 3E	403		LDA	A2L	
FDCD:90 02 FDD1	404		BCC	ADD	
FDCF:49 FF	405		EOR	#\$FF	;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C	406	ADD	ADC	A1L	

FDD3:48		407	PHA		
FDD4:A9 BD		408	LDA #\$BD		;PRINT '=', THEN RESULT
FDD6:20 ED FD		409	JSR COUT		
FDD9:68		410	PLA		
FDDA:48		411 PRBYTE	PHA		;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A		412	LSR A		; (DESTROYS A-REG)
FDDC:4A		413	LSR A		
FDDD:4A		414	LSR A		
FDDE:4A		415	LSR A		
FDDF:20 E5 FD		416	JSR PRHEXZ		
FDE2:68		417	PLA		
FDE3:29 0F		418 PRHEX	AND #\$0F		;PRINT HEX DIGIT IN A-REG
FDE5:09 B0		419 PRHEXZ	ORA #\$B0		;LSBITS ONLY.
FDE7:C9 BA		420	CMP #\$BA		
FDE9:90 02 FDED		421	BCC COUT		
FDEB:69 06		422	ADC #\$06		
FDED:		423 *			
FDED:6C 36 00		424 COUT	JMP (CSWL)		;VECTOR TO USER OUTPUT ROUTINE
FDF0:		425 *			
FDF0:48		426 COUT1	PHA		;save original character
FDF1:C9 A0		427	CMP #\$A0		;is it a control?
FDF3:4C 95 FC		428	JMP DOCOUT1		;=>mask if not; return to COUTZ1
FDF6:		429 *			
FDF6:48		430 COUTZ	PHA		;save original character
FDF7:84 35		431 COUTZ1	STY YSAV1		;save Y
FDF9:A8		432	TAY		;save masked character
FDFA:68		433	PLA		;get original char
FDFB:4C 46 FC		434	JMP NEWVW		;new entry to vidwait
FDFE:EA		435	NOP		
FDFE:EA		436	NOP		
FE00:		437 *			
FE00:C6 34		438 BL1	DEC YSAV		
FE02:F0 9F FDA3		439	BEQ XAM8		
FE04:CA		440 BLANK	DEX		;BLANK TO MON
FE05:D0 16 FE1D		441	BNE SETMDZ		;AFTER BLANK
FE07:C9 BA		442	CMP #\$BA		;DATA STORE MODE?
FE09:D0 BB FDC6		443	BNE XAMPM		; NO; XAM, ADD, OR SUBTRACT.
FE0B:85 31		444 STOR	STA MODE		;KEEP IN STORE MODE
FE0D:A5 3E		445	LDA A2L		
FE0F:91 40		446	STA (A3L),Y		;STORE AS LOW BYTE AT (A3)
FE11:E6 40		447	INC A3L		
FE13:D0 02 FE17		448	BNE RTS5		;INCR A3, RETURN.
FE15:E6 41		449	INC A3H		
FE17:60		450 RTS5	RTS		
FE18:		451 *			
FE18:A4 34		452 SETMODE	LDY YSAV		;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01		453	LDA IN-1,Y		; '-', '.' AS MODE
FE1D:85 31		454 SETMDZ	STA MODE		
FE1F:60		455	RTS		
FE20:		456 *			
FE20:A2 01		457 LT	LDX #\$01		
FE22:B5 3E		458 LT2	LDA A2L,X		;COPY A2 (2 BYTES) TO
FE24:95 42		459	STA A4L,X		; A4 AND A5
FE26:95 44		460	STA A5L,X		

FE28:CA		461	DEX	
FE29:10 F7	FE22	462	BPL	LT2
FE2B:60		463	RTS	
FE2C:		464 *		
FE2C:B1 3C		465	MOVE	LDA (A1L),Y ;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42		466		STA (A4L),Y
FE30:20 B4 FC		467		JSR NXTA4
FE33:90 F7	FE2C	468	BCC	MOVE
FE35:60		469	RTS	
FE36:		470 *		
FE36:B1 3C		471	VFY	LDA (A1L),Y ;VERIFY (A1) THRU (A2)
FE38:D1 42		472		CMP (A4L),Y ; WITH (A4)
FE3A:F0 1C	FE58	473		BEQ VFYOK
FE3C:20 92 FD		474		JSR PRA1
FE3F:B1 3C		475		LDA (A1L),Y
FE41:20 DA FD		476		JSR PRBYTE
FE44:A9 A0		477		LDA #\$A0
FE46:20 ED FD		478		JSR COUT
FE49:A9 A8		479		LDA #\$A8
FE4B:20 ED FD		480		JSR COUT
FE4E:B1 42		481		LDA (A4L),Y
FE50:20 DA FD		482		JSR PRBYTE
FE53:A9 A9		483		LDA #\$A9
FE55:20 ED FD		484		JSR COUT
FE58:20 B4 FC		485	VFYOK	JSR NXTA4
FE5B:90 D9	FE36	486		BCC VFY
FE5D:60		487	RTS	
FE5E:		488 *		
FE5E:20 75 FE		489	LIST	JSR A1PC ;MOVE A1 (2 BYTES) TO
FE61:A9 14		490		LDA #\$14 ; PC IF SPEC'D AND
FE63:48		491	LIST2	PHA ; DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 D0 F8		492		JSR INSTDSP
FE67:20 53 F9		493		JSR PCADJ ;ADJUST PC AFTER EACH INSTRUCTION.
FE6A:85 3A		494		STA PCL
FE6C:84 3B		495		STY PCH
FE6E:68		496		PLA
FE6F:38		497		SEC
FE70:E9 01		498		SBC #\$01 ;NEXT OF 20 INSTRUCTIONS
FE72:D0 EF	FE63	499		BNE LIST2
FE74:60		500	RTS	
FE75:		501 *		
FE75:8A		502	A1PC	TXA ;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07	FE7F	503		BEQ A1PCRTS ; COPY IT FROM A1 TO PC.
FE78:B5 3C		504	A1PCLP	LDA A1L,X ;YEP, SO COPY IT.
FE7A:95 3A		505		STA PCL,X
FE7C:CA		506		DEX
FE7D:10 F9	FE78	507		BPL A1PCLP
FE7F:60		508	A1PCRTS	RTS
FE80:		509 *		
FE80:A0 3F		510	SETINV	LDY #\$3F ;SET FOR INVERSE VID
FE82:D0 02	FE86	511		BNE SETIFLG ; VIA COUT1
FE84:A0 FF		512	SETNORM	LDY #\$FF ;SET FOR NORMAL VID
FE86:84 32		513	SETIFLG	STY INVFLG
FE88:60		514	RTS	

```

FE89:          515 *
FE89:A9 00     516 SETKBD LDA #$00          ;DO 'IN#O'
FE8B:85 3E     517 INPORT STA A2L          ;DO 'IN#AREG'
FE8D:A2 38     518 INPRT LDX #KSWL
FE8F:A0 18     519          LDY #KEYIN
FE91:D0 08     520          BNE IOPRT
FE93:          521 *
FE93:A9 00     522 SETVID LDA #$00          ;DO 'PR#O'
FE95:85 3E     523 OUTPORT STA A2L          ;DO 'PR#AREG'
FE97:A2 36     524 OUTPRT LDX #CSWL
FE99:A0 F0     525          LDY #COUT1
FE9B:A5 3E     526 IOPRT LDA A2L          ;SET INPUT/OUTPUT VECTORS
FE9D:29 0F     527          AND #$0F
FE9F:F0 04     528          BEQ IOPRT1
FEA1:09 C0     529          ORA #<IOADR
FEA3:A0 00     530          LDY #$00
FEA5:94 00     531 IOPRT1 STY LOCO,X          ;save low byte of hook
FEA7:95 01     532          STA LOCL,X          ;save acc
FEA9:A0 0E     533          LDY #$E          ;code=PR# /IN#
FEAB:4C B4 FB  534 GOTOCX4 JMP GOTOCX          ;perform call
FEAE:          535 *
FEAE:EA       536          NOP
FEAF:00       537 CKSUMFIX DFB 0          ;/RRA0981
FEB0:          538 *          ;-->CORRECT CKSUM AT CREATE TIME.
FEB0:4C 00 E0  539 XBASIC JMP BASIC          ;TO BASIC, COLD START
FEB3:4C 03 E0  540 BASCONT JMP BASIC2          ;TO BASIC, WARM START
FEB6:20 75 FE  541 GO          JSR AIPC          ;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF  542          JSR RESTORE          ;RESTORE FAKE REGISTERS
FEBE:6C 3A 00  543          JMP (PCL)          ;AND GO!
FEBF:4C D7 FA  544 REGZ          JMP REGDSP          ;GO DISPLAY REGISTERS
FEC2:60       545 TRACE          RTS          ;TRACE IS GONE
FEC3:EA       546          NOP
FEC4:60       547 STEPZ          RTS          ;STEP IS GONE
FEC5:          548 *
FEC5:          549 * Return here from GOTOCX
FEC5:          550 *
FEC5:          551 * NOTE: This address is hard-coded in BFUNC of the
FEC5:          552 * video firmware
FEC5:          553 *
FEC5:8D 06 C0  554 RETCX1 STA SETSLOTXROM ;restore bank
FEC8:60       555 RETCX2 RTS          ;simply return
FEC9:EA       556          NOP
FECA:          557 *
FECA:4C F8 03  558 USR          JMP USRADR          ;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:          559 *
FECD:A9 40     560 WRITE          LDA #$40
FECF:8D 07 C0  561 WRT2          STA SETINTCXROM ;set internal ROM
FED2:20 AA C5  562          JSR WRITE2          ;write to tape
FED5:F0 2C FF03 563          BEQ RD2          ;=>always set slots, beep
FED7:          564 *
FED7:          565 * SEARCH is called with a Monitor command of the form
FED7:          566 * HHL<ADR1.ADR2 in which ADR1 < ADR2 and LL precedes HH
FED7:          567 * in memory. If HH is 0, or omitted (LL<ADR1.ADR2), then
FED7:          568 * the single byte LL is searched for. You cannot search for

```

```

FED7:          569 * a two byte pair with a high byte of 0. A list of all
FED7:          570 * addresses containing the specified pattern is displayed.
FED7:          571 *
FED7:A0 01     572 SEARCH LDY #1          ;set Y to 1
FED9:A5 43     573 LDA A4H          ;is high byte 0?
FEDB:F0 04 FEE1 574 BEQ SRCH1        ;=>yes, only look for low byte
FEDD:D1 3C     575 CMP (A1L),Y      ;check high byte first
FEDF:D0 0A FEEB 576 BNE SRCH2        ;=>no match, try next byte
FEE1:88       577 SRCH1 DEY          ;match, now check low byte
FEE2:A5 42     578 LDA A4L          ;get low byte
FEE4:D1 3C     579 CMP (A1L),Y      ;does it match?
FEE6:D0 03 FEEB 580 BNE SRCH2        ;=>no match, try next byte
FEE8:20 92 FD 581 JSR PRA1          ;bytes match, print address
FEEB:20 BA FC 582 SRCH2 JSR NXTA1      ;increment address
FEEE:90 E7 FED7 583 BCC SEARCH      ;set Y back to 1
FEFO:60       584 RTS
FEF1:         585 *
FEF1:A0 0D     586 MINI LDY #$D          ;dispatch mini-assembler call to
FEF3:20 B4 FB 587 JSR GOTOCX        ;get internal ROM switched in
FEF6:         588 *
FEF6:20 00 FE 589 CRMON JSR BL1          ;HANDLE CR AS BLANK
FEF9:68       590 PLA              ; THEN POP STACK
FEFA:68       591 PLA              ; AND RETURN TO MON
FEFB:D0 6C FF69 592 BNE MONZ        ;(ALWAYS)
FEFD:         593 *
FEFD:8D 07 C0 594 READ STA SETINTCXROM ;set internal ROM
FF00:20 D1 C5 595 JSR XREAD          ;do tape read
FF03:8D 06 C0 596 RD2 STA SETSLOT CXROM ;restore slot CX
FF06:F0 32 FF3A 597 BEQ BELL          ;read (write) ok, beep
FF08:D0 23 FF2D 598 BNE PRERR        ;error, print message
FF0A:         599 *
FF0A:C1 F0 F0 EC 600 TITLE ASC "Apple  //"
FF13:         601 *
FF13:         602 * NNBL gets the next non-blank for the mini-assembler
FF13:         603 *
FF13:20 FD FC 604 NNBL JSR UPMON        ;get char, upshift, INY
FF16:C9 A0     605 CMP #$A0          ;is it blank?
FF18:F0 F9 FF13 606 BEQ NNBL        ;yes, keep looking
FF1A:60       607 RTS
FF1B:         608 *
FF1B:B0 6D FF8A 609 LOOKASC BCS DIG          ;it was a digit
FF1D:C9 A0     610 CMP #$A0          ;check for quote (')
FF1F:D0 28 FF49 611 BNE RTS6          ;nope, return char
FF21:B9 00 02 612 LDA $200,Y        ;else get next char
FF24:A2 07     613 LDX #7          ;for shifting asc into A2L and A2H
FF26:C9 8D     614 CMP #$8D          ;was it CR?
FF28:F0 7D FFA7 615 BEQ GETNUM        ;yes, go handle CR
FF2A:C8       616 INY              ;advance index
FF2B:D0 63 FF90 617 BNE NXTBIT        ;=>(always) into A2L and A2H
FF2D:         618 *
FF2D:A9 C5     619 PRERR LDA #$C5        ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD 620 JSR COUT          ; FWEEPER.
FF32:A9 D2     621 LDA #$D2
FF34:20 ED FD 622 JSR COUT

```

FF37:20 ED FD	623	JSR	COUT	
FF3A:	624 *			
FF3A:A9 87	625 BELL	LDA	#\$87	;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD	626	JMP	COUT	
FF3F:	627 *			
FF3F:A5 48	628 RESTORE	LDA	STATUS	;RESTORE 6502 REGISTER CONTENTS
FF41:48	629	PHA		; USED BY DEBUG SOFTWARE
FF42:A5 45	630	LDA	A5H	
FF44:A6 46	631 RESTRI	LDX	XREG	
FF46:A4 47	632	LDY	YREG	
FF48:28	633	PLP		
FF49:60	634 RTS6	RTS		
FF4A:	635 *			
FF4A:85 45	636 SAVE	STA	A5H	;SAVE 6502 REGISTER CONTENTS
FF4C:86 46	637 SAV1	STX	XREG	; FOR DEBUG SOFTWARE
FF4E:84 47	638	STY	YREG	
FF50:08	639	PHP		
FF51:68	640	PLA		
FF52:85 48	641	STA	STATUS	
FF54:BA	642	TSX		
FF55:86 49	643	STX	SPNT	
FF57:D8	644	CLD		
FF58:60	645	RTS		
FF59:	646 *			
FF59:20 84 FE	647 OLDRST	JSR	SETNORM	;SET SCREEN MODE
FF5C:20 2F FB	648	JSR	INIT	; AND INIT KBD/SCREEN
FF5F:20 93 FE	649	JSR	SETVID	; AS I/O DEVS.
FF62:20 89 FE	650	JSR	SETKBD	
FF65:	651 *			
FF65:D8	652 MON	CLD		;MUST SET HEX MODE!
FF66:20 3A FF	653	JSR	BELL	;FWEEPER.
FF69:A9 AA	654 MONZ	LDA	#\$AA	; '*' PROMPT FOR MONITOR
FF6B:85 33	655	STA	PROMPT	
FF6D:20 67 FD	656	JSR	GETLNZ	;READ A LINE OF INPUT
FF70:20 C7 FF	657	JSR	ZMODE	;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF	658 NXTITM	JSR	GETNUM	;GET ITEM, NON-HEX
FF76:84 34	659	STY	YSAV	; CHAR IN A-REG.
FF78:A0 17	660	LDY	#\$17	; X-REG=0 IF NO HEX INPUT
FF7A:88	661 CHRSRCH	DEY		
FF7B:30 E8 FF65	662	BMI	MON	;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CC FF	663	CMP	CHRTBL,Y	;FIND COMMAND CHAR IN TABLE
FF80:D0 F8 FF7A	664	BNE	CHRSRCH	;NOT THIS TIME
FF82:20 BE FF	665	JSR	TOSUB	;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34	666	LDY	YSAV	;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF	667	JMP	NXTITM	
FF8A:	668 *			
FF8A:A2 03	669 DIG	LDX	#\$03	
FF8C:0A	670	ASL	A	
FF8D:0A	671	ASL	A	;GOT HEX DIGIT,
FF8E:0A	672	ASL	A	; SHIFT INTO A2
FF8F:0A	673	ASL	A	
FF90:0A	674 NXTBIT	ASL	A	
FF91:26 3E	675	ROL	A2L	
FF93:26 3F	676	ROL	A2H	

FF95:CA		677	DEX		;LEAVE X=\$FF IF DIG
FF96:10 F8	FF90	678	BPL	NXTBIT	
FF98:A5 31		679	LDA	MODE	
FF9A:D0 06	FFA2	680	BNE	NXTBS2	;IF MODE IS ZERO,
FF9C:B5 3F		681	LDA	A2H,X	; THEN COPY A2 TO A1 AND A3
FF9E:95 3D		682	STA	A1H,X	
FFA0:95 41		683	STA	A3H,X	
FFA2:E8		684	NXTBS2	INX	
FFA3:F0 F3	FF98	685	BEQ	NXTBAS	
FFA5:D0 06	FFAD	686	BNE	NXTCHR	
FFA7:		687	*		
FFA7:A2 00		688	GETNUM	LDX #\$00	;CLEAR A2
FFA9:86 3E		689	STX	A2L	
FFAB:86 3F		690	STX	A2H	
FFAD:20 FD FC		691	NXTCHR	JSR UPMON	;get char, upshift, INY
FFB0:EA		692	NOP		;INY now done in UPMON
FFB1:49 B0		693	EOR	#\$B0	
FFB3:C9 0A		694	CMP	#\$0A	
FFB5:90 D3	FF8A	695	BCC	DIG	;BR IF HEX DIGIT
FFB7:69 88		696	ADC	#\$88	
FFB9:C9 FA		697	CMP	#\$FA	
FFBB:4C 1B FF		698	JMP	LOOKASC	;check for ASCII input
FFBE:		699	*		
FFBE:A9 FE		700	TOSUB	LDA #<GO	;DISPATCH TO SUBROUTINE, BY
FFC0:48		701	PHA		; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E3 FF		702	LDA	SUBTBL,Y	; THEN THE LO-ORDER SUBR ADDR
FFC4:48		703	PHA		; ONTO THE STACK,
FFC5:A5 31		704	LDA	MODE	; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00		705	ZMODE	LDY #\$00	; MODE IN A-REG),
FFC9:84 31		706	STY	MODE	
FFCB:60		707	RTS		; AND 'RTS' TO THE SUBROUTINE!
FFCC:		708	*		
FFCC:BC		709	CHRTBL	DFB \$BC	;^C (BASIC WARM START)
FFCD:B2		710	DFB	\$B2	;^Y (USER VECTOR)
FFCE:BE		711	DFB	\$BE	;^E (OPEN AND DISPLAY REGISTERS)
FFCF:9A		712	DFB	\$9A	;! (enter mini-assembler)
FFD0:EF		713	DFB	\$EF	;V (MEMORY VERIFY)
FFD1:C4		714	DFB	\$C4	;^K (IN#SLOT)
FFD2:EC		715	DFB	\$EC	;S (search for 2 bytes)
FFD3:A9		716	DFB	\$A9	;^P (PR#SLOT)
FFD4:BB		717	DFB	\$BB	;^B (BASIC COLD START)
FFD5:A6		718	DFB	\$A6	; '-' (SUBTRACTION)
FFD6:A4		719	DFB	\$A4	; '+' (ADDITION)
FFD7:06		720	DFB	\$06	;M (MEMORY MOVE)
FFD8:95		721	DFB	\$95	; '<' (DELIMITER FOR MOVE, VFY)
FFD9:07		722	DFB	\$07	;N (SET NORMAL VIDEO)
FFDA:02		723	DFB	\$02	;I (SET INVERSE VIDEO)
FFDB:05		724	DFB	\$05	;L (DISASSEMBLE 20 INSTRS)
FFDC:F0		725	DFB	\$F0	;W (WRITE TO TAPE)
FFDD:00		726	DFB	\$00	;G (EXECUTE PROGRAM)
FFDE:EB		727	DFB	\$EB	;R (READ FROM TAPE)
FFDF:93		728	DFB	\$93	; ':' (MEMORY FILL)
FFEO:A7		729	DFB	\$A7	; '.' (ADDRESS DELIMITER)
FFE1:C6		730	DFB	\$C6	; 'CR' (END OF INPUT)


```

FFE2:99      731      DFB $99      ;BLANK
FFE3:        732 *
FFE3:        733 * Table of low order monitor routine dispatch
FFE3:        734 * addresses. High byte always $FE
FFE3:        735 *
FFE3:B2      736 SUBTBL DFB >BASCONT-1 ;^C (BASIC warm start)
FFE4:C9      737      DFB >USR-1 ;^Y (not used)
FFE5:BE      738      DFB >REGZ-1 ;^E (open and display registers)
FFE6:F0      739      DFB >MINI-1 ;mini assembler
FFE7:35      740      DFB >Vfy-1 ;V (memory verify)
FFE8:8C      741      DFB >INPRT-1 ;^K (IN#SLOT)
FFE9:D6      742      DFB >SEARCH-1 ;search for pattern
FFEA:96      743      DFB >OUTPRT-1 ;^P (PR#SLOT)
FFEB:AF      744      DFB >XBASIC-1 ;^B (BASIC cold start)
FFEC:17      745      DFB >SETMODE-1 ;'-' (subtraction)
FFED:17      746      DFB >SETMODE-1 ;'+' (addition)
FFEE:2B      747      DFB >MOVE-1 ;M (memory move)
FFEF:1F      748      DFB >LT-1 ;'^' (delim for move,vfy)
FFF0:83      749      DFB >SETNORM-1 ;N (set normal video)
FFF1:7F      750      DFB >SETINV-1 ;I (set inverse video)
FFF2:5D      751      DFB >LIST-1 ;L (disassemble 20 instrs)
FFF3:CC      752      DFB >WRITE-1 ;W (write to tape)
FFF4:B5      753      DFB >GO-1 ;G (execute program)
FFF5:FC      754      DFB >READ-1 ;R (read from tape)
FFF6:17      755      DFB >SETMODE-1 ;':' (memory fill)
FFF7:17      756      DFB >SETMODE-1 ;'.' (address delimiter)
FFF8:F5      757      DFB >CRMON-1 ;'CR' (end of input)
FFF9:03      758      DFB >BLANK-1 ;BLANK
FFFA:        759 *
FFFA:FB 03   760      DW NMI ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA   761      DW RESET ;RESET VECTOR
FFFE:FA C3   762      DW IRQ ;INTERRUPT REQUEST VECTOR
0000:        19      INCLUDE MINI
0000:        1 *
0000:        2 * Apple //e Mini Assembler
0000:        3 *
0000:        4 * Got mnemonic, check address mode
0000:        5 *
C4C8: C4C8   6      ORG C3ORG+$1C8
C4C8:        7 *
C4C8:20 13 FF 8 AMOD1 JSR NNBL ;get next non-blank
C4CB:84 34   9      STY YSAV ;save Y
C4CD:DD B4 F9 10     CMP CHAR1,X
C4D0:D0 13 C4E5 11     BNE AMOD2
C4D2:20 13 FF 12     JSR NNBL ;get next non-blank
C4D5:DD BA F9 13     CMP CHAR2,X
C4D8:F0 0D C4E7 14     BEQ AMOD3
C4DA:BD BA F9 15     LDA CHAR2,X ;done yet?
C4DD:F0 07 C4E6 16     BEQ AMOD4
C4DF:C9 A4 17     CMP #$A4 ;if "$" then done
C4E1:F0 03 C4E6 18     BEQ AMOD4
C4E3:A4 34 19     LDY YSAV ;restore Y
C4E5:18 20 AMOD2 CLC
C4E6:88 21 AMOD4 DEY

```

```

C4E7:26 44      22 AMOD3   ROL   A5L      ;shift bit into format
C4E9:E0 03      23         CPX   #$03
C4EB:D0 0D      24         BNE   AMOD6
C4ED:20 A7 FF   25         JSR   GETNUM
C4F0:A5 3F      26         LDA   A2H      ;get high byte of address
C4F2:F0 01      27         BEQ   AMOD5    ;=>
C4F4:E8         28         INX
C4F5:86 35      29 AMOD5   STX   YSAV1
C4F7:A2 03      30         LDX   #$03
C4F9:88         31         DEY
C4FA:86 3D      32 AMOD6   STX   A1H
C4FC:CA         33         DEX
C4FD:10 C9      34         BPL   AMOD1
C4FF:60         35         RTS
C500:         36 *
CF3A:         CF3A 37         ORG   C8ORG+$73A
CF3A:         38 *
CF3A:         39 * Calculate offset byte for relative addresses
CF3A:         40 *
CF3A:E9 81      41 REL     SBC   #$81      ;calc relative address
CF3C:4A         42         LSR   A
CF3D:D0 14      43         BNE   GOERR     ;bad branch
CF3F:A4 3F      44         LDY   A2H
CF41:A6 3E      45         LDX   A2L
CF43:D0 01      46         BNE   REL1
CF45:88         47         DEY      ;point to offset
CF46:CA         48 REL1    DEX      ;displacement - 1
CF47:8A         49         TXA
CF48:18         50         CLC
CF49:E5 3A      51         SBC   PCL      ;subtract current PCL
CF4B:85 3E      52         STA   A2L      ;and save as displacement
CF4D:10 01      53         BPL   REL2    ;check page
CF4F:C8         54         INY
CF50:98         55 REL2    TYA      ;get page
CF51:E5 3B      56         SBC   PCH      ;check page
CF53:D0 40      57 GOERR   BNE   MINIERR ;display error
CF55:         58 *
CF55:         59 * Move instruction to memory
CF55:         60 *
CF55:A4 2F      61 MOVINST LDY   LENGTH ;get instruction length
CF57:B9 3D 00    62 MOV1    LDA   A1H,Y   ;get a byte
CF5A:91 3A      63         STA   (PCL),Y ;and move it
CF5C:88         64         DEY
CF5D:10 F8      65         BPL   MOV1
CF5F:         66 *
CF5F:         67 * Display instruction
CF5F:         68 *
CF5F:20 48 F9    69         JSR   PRBLNK   ;print blanks to make ProDOS work
CF62:20 1A FC    70         JSR   UP      ;move up 2 lines
CF65:20 1A FC    71         JSR   UP
CF68:4C E3 FC    72         JMP   DISLIN   ;disassemble it, =>DOINST
CF6B:         73 *
CF6B:         74 * Compare disassembly of all known opcodes with
CF6B:         75 * the one typed in until a match is found

```

```

CF6B:          76 *
CF6B:A5 3D      77 GETOP   LDA  A1H      ;get opcode
CF6D:20 8E F8    78         JSR  INSDS2   ;determine mnemonic index
CF70:AA          79         TAX           ;X = index
CF71:BD 00 FA     80         LDA  MNEMR,X  ;get right half of index
CF74:C5 42        81         CMP  A4L      ;does it match entry?
CF76:D0 13      CF8B  82         BNE  NXTOP   ;=>try next opcode
CF78:BD C0 F9     83         LDA  MNEML,X  ;get left half of index
CF7B:C5 43        84         CMP  A4H      ;does it match entry?
CF7D:D0 0C      CF8B  85         BNE  NXTOP   ;=>no, try next opcode
CF7F:C5 44        86         LDA  A5L      ;found opcode, check address mode
CF81:A4 2E        87         LDY  FORMAT   ;get addr. mode format for that opcode
CF83:C0 9D        88         CPY  #$9D     ;is it relative?
CF85:F0 B3      CF3A  89         BEQ  REL      ;=>yes, calc relative address
CF87:C5 2E        90         CMP  FORMAT   ;does mode match?
CF89:F0 CA      CF55  91         BEQ  MOVINST  ;=>yes, move instruction to memory
CF8B:C6 3D        92 NXTOP   DEC  A1H      ;else try next opcode
CF8D:D0 DC      CF6B  93         BNE  GETOP   ;=>go try it
CF8F:E6 44        94         INC  A5L      ;else try next format
CF91:C6 35        95         DEC  YSAV1    ;
CF93:F0 D6      CF6B  96         BEQ  GETOP   ;=>go try next format
CF95:          97 *
CF95:          98 * Point to the error with a caret, beep, and fall
CF95:          99 * into the mini-assembler.
CF95:         100 *
CF95:A4 34      101 MINIERR LDY  YSAV      ;get position
CF97:98         102 ERR2   TYA
CF98:AA         103         TAX
CF99:4C D2 FC    104         JMP  ERR3      ;display error, =>DOINST
CF9C:          105 *
CF9C:          106 * Read a line of input. If prefaced with " ", decode
CF9C:          107 * mnemonic. If "$" do monitor command. Otherwise parse
CF9C:          108 * hex address before decoding mnemonic.
CF9C:          109 *
CF9C:20 C7 FF    110 DOINST  JSR  ZMODE     ;clear mode
CF9F:AD 00 02    111         LDA  $200     ;get first char in line
CFA2:C9 A0      112         CMP  #$A0     ;if blank,
CFA4:F0 12      CFB8  113         BEQ  DOLIN   ;=>go attempt disassembly
CFA6:C9 8D      114         CMP  #$8D     ;is it return?
CFA8:D0 01      CFAB  115         BNE  GETI1   ;=>no, continue
CFAA:60         116         RTS          ;else return to Monitor
CFAB:          117 *
CFAB:20 A7 FF    118 GETI1   JSR  GETNUM    ;parse hexadecimal input
CFAE:C9 93      119         CMP  #$93     ;look for "ADDR:"
CFB0:D0 E5      CF97  120 GOERR2 BNE  ERR2   ;no ":", display error
CFB2:8A         121         TXA          ;X nonzero if address entered
CFB3:F0 E2      CF97  122         BEQ  ERR2   ;no "ADDR", display error
CFB5:          123 *
CFB5:20 78 FE    124         JSR  ALPCLP    ;move address to PC
CFB8:A9 03      125 DOLIN   LDA  #$03     ;get starting opcode
CFBA:85 3D      126         STA  A1H      ;and save
CFBC:20 13 FF    127 NXTCH  JSR  NNBL     ;get next non-blank
CFBF:0A         128         ASL  A        ;validate entry
CFC0:E9 BE      129         SBC  #$BE

```

CFC2:C9 C2	130	CMP	#\$C2	
CFC4:90 D1 CF97	131	BCC	ERR2	;=>flag bad mnemonic
CFC6:	132 *			
CFC6:	133 *	Form mnemonic for later comparison		
CFC6:	134 *			
CFC6:0A	135	ASL	A	
CFC7:0A	136	ASL	A	
CFC8:A2 04	137	LDX	#\$04	
CFCA:0A	138	NXTMN	ASL A	
CFCB:26 42	139	ROL	A4L	
CFCD:26 43	140	ROL	A4H	
CFCF:CA	141	DEX		
CFD0:10 F8 CFCA	142	BPL	NXTMN	
CFD2:C6 3D	143	DEC	A1H	;decrement mnemonic count
CFD4:F0 F4 CFCA	144	BEQ	NXTMN	
CFD6:10 E4 CFBC	145	BPL	NXTCH	
CFD8:A2 05	146	LDX	#\$5	;index into address mode tables
CFDA:20 C8 C4	147	JSR	AMOD1	;do this elsewhere
CFDD:A5 44	148	LDA	A5L	;get format
CFDF:0A	149	ASL	A	
CFE0:0A	150	ASL	A	
CFE1:05 35	151	ORA	YSAV1	
CFE3:C9 20	152	CMP	#\$20	
CFE5:B0 06 CFED	153	BCS	AMOD7	
CFE7:A6 35	154	LDX	YSAV1	;get our format
CFE9:F0 02 CFED	155	BEQ	AMOD7	
CFEB:09 80	156	ORA	#\$80	
CFED:85 44	157	AMOD7	STA A5L	;update format
CFEF:84 34	158	STY	YSAV	;update position
CFF1:B9 00 02	159	LDA	\$0200,Y	;get next character
CFF4:C9 BB	160	CMP	#\$BB	;is it a " ;"?
CFF6:F0 04 CFFC	161	BEQ	AMOD8	;=>yes, skip comment
CFF8:C9 8D	162	CMP	#\$8D	;is it carriage return
CFFA:D0 B4 CFEB	163	BNE	GOERR2	
CFFC:4C 6B CF	164	AMOD8	JMP GETOP	;get next opcode
CFFF:	165 *			
CFFF:00	166	DFB	\$00	;byte for making CTOD checksum ok

Glossary

accumulator: The register in the 65C02 microprocessor where most computations are performed.

ACIA: Acronym for *Asynchronous Communications Interface Adapter*. The ACIA is a chip that converts data from parallel to serial form and vice versa. Its internal registers control and keep track of the sending and receiving of data. Firmware and software set and change the status of these internal registers.

acronym: A word formed from the initial letters of a name or phrase, such as *ROM*, from *read-only memory*.

address: A number that specifies a single byte of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from \$0000 to \$FFFF (in hexadecimal).

algorithm: A step-by-step procedure for solving a problem or accomplishing a task.

analog: Represented in terms of a physical quantity that can vary smoothly and continuously over a

range of values. For example, a conventional 12-hour clock face is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

analog data: Data in the form of continuously variable physical quantities. Compare **digital data**.

analog signal: A signal that varies continuously over time.

analog-to-digital converter: A device that converts quantities from analog to digital form. For example, hand controls used on Apple II family computers convert the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes abruptly even when the dial is turned smoothly.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR**, **exclusive OR**, **NOT**.

ANSI: Acronym for *American National Standards Institute*, which sets standards for many fields and is the most common standard for terminals.

Apple IIc: A transportable personal computer in the Apple II family, with a disk drive and 80-column capability built in.

Apple IIe: A personal computer in the Apple II family.

Apple IIe 80-Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary slot and converts the computer's display of text from 40-column width to 80-column width.

Apple IIe Extended 80-Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary slot and converts the computer's display of text from 40-column width to 80-column width while extending its memory capacity by 64K bytes.

Apple II Pascal: A software system that lets you create and execute programs written in the Pascal programming language, adapted by Apple Computer from the UCSD (University of California, San Diego) Pascal Operating System and sold for use with the Apple II family of computers.

Applesoft BASIC: An extended version of the BASIC programming language used with the Apple II family of computers. An interpreter for creating and executing programs in Applesoft is built into the computer's firmware. Compare **Integer BASIC**.

application program: A program that puts the resources and capabilities of the computer to use for some specific purpose or task, such as word processing, data base management, or graphics. Compare **system program**.

argument: The value on which a function operates.

arithmetic expression: A combination of numbers and arithmetic operators (such as $3 + 5$) that indicates some operation to be carried out.

arithmetic operator: An operator, such as $+$, that combines numeric values to produce a numeric result. Compare **relational operator**, **logical operator**.

ASCII: Acronym for *American Standard Code for Information Interchange*, pronounced *ASK ee*. A code in which the numbers from 0

to 127 stand for text characters—including the letters of the alphabet, the digits 0 through 9, punctuation marks, special characters, and control characters—used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

asserted: Made true (positive in positive-true logic; negative in negative-true logic).

asynchronous transmission: Not synchronized by or with a clocking signal. Transmission in which each information character is individually synchronized, usually by the use of start and stop bits. The gap between each character isn't necessarily fixed. Compare **synchronous transmission**.

auxiliary slot: The special expansion slot inside the Apple IIe used for the Apple 80-Column Text Card or Extended 80-Column Text Card.

base address: In indexed addressing, the fixed component of an address.

BASIC: Acronym for *Beginner's All-purpose Symbolic Instruction Code*. A high-level programming language designed to be easy to learn and use. Two versions of BASIC are available from Apple Computer for use with all Apple II family systems: Applesoft (built into firmware) and Integer BASIC (provided on the *ProDOS User's Disk*).

baud: Unit of signaling speed taken from the name Baudot. The speed in bauds is equal to the number of discrete conditions or signal events per second regardless of the information content of those signals. Often equated (though not precisely) with bits per second. Compare **bit rate**.

binary: The representation of numbers in terms of powers of two, using the two digits 0 and 1. Commonly used in computers because the values 0 and 1 can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen. A single binary digit—a 0 or a 1—is called a **bit**.

binary digit: The smallest unit of information in the binary number system. Also called a **bit**.

binary operator: An operator that combines two operands to produce a result; for example, **+** is a binary arithmetic operator, **<** is a binary relational operator, and **OR** is a binary logical operator. Compare **unary operator**.

bit: The smallest item of useful information a computer can handle. Usually represented as a 1 or a 0. Eight bits equal one byte.

bit rate: The speed at which bits are transmitted, usually expressed as **bps** or **bits per second**. Compare **baud**.

board: See **printed-circuit board**.

body: The statements or instructions that make up a part of a program, such as a loop or a subroutine.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to *pull itself up by its own bootstraps*—hence the term *bootstrapping* or *booting*.

boot disk: See **startup disk**.

bootstrap: See **boot**.

bps: See **bit rate**.

branch: To send program execution to a line or statement other than the next in sequence.

BREAK: A SPACE (0) signal, sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a time-sharing service.

BRK: An instruction that causes the 65C02 microprocessor to halt.

buffer: A memory area that holds information until it can be processed.

bug: An error in a program that causes it not to work as intended.

bus: A group of wires that transmit related information from one part of a computer system to another.

byte: A sequence of eight bits that represents an instruction, a letter, a number, or a punctuation mark.

cable: A group of wires used to carry information between two devices. How many wires are used varies with the type of connection.

call: To request the execution of a subroutine or function.

card: See **peripheral card**.

carriage return: An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the subsequent character on the left margin.

carrier: The background signal on a communication channel that is modified to *carry* the information. Under RS232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

carry flag: A status bit in the 65C02 microprocessor, used to hold the high-order bit (the *carry* bit) in addition and subtraction.

central processing unit: See **processor**.

character: Any symbol that has a widely understood meaning. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer. Others are used to control various functions of the computer. See **control character**.

character code: A number used to represent a text character for processing by a computer system.

character set: The entire set of characters that can be either shown on a monitor or used to code computer instruction. In a printer, the entire set of characters that the printer is capable of printing.

circuit board: A collection of integrated circuits (chips) on a board.

Clear To Send: An RS232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

cold start: The process of starting up the Apple II when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, then loading and running a program.

column: A vertical arrangement of graphics points or character spaces on the monitor screen.

command: A word or character that causes the computer to do something.

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare **interpreter**.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it.

computer: An electronic device that performs predefined (programmed) computations at high speed and with great accuracy. A machine that is used to store, transfer, and transform information.

computer language: See **programming language**.

computer system: A computer and its associated hardware, firmware, and software.

conditional branch: A branch that depends on the truth of a condition or the value of an expression. Compare **unconditional branch**.

configuration: The hardware and software arrangement of a system.

connector: A physical device such as a plug, socket, or jack, used to connect two devices to one another.

console: The Apple IIe's video display and keyboard together make up the console. This is the part of the Apple IIe you communicate with directly.

constant: A symbol in a program that represents a fixed, unchanging value. Compare **variable**.

CONTROL: A key that when pressed in conjunction with another key makes that other key behave differently.

CONTROL-RESET: This combination of keystrokes usually causes an AppleSoft program or command to stop immediately. If a program disables the **CONTROL-RESET** feature, you need to turn the computer off to get the program to stop.

control character: A non-printing character that controls or modifies the way information is printed or displayed. Control characters have ASCII values between 0 and 31, and are typed from a keyboard by holding down **CONTROL** while pressing some other key. For example, the character Control-M (ASCII code 13) means “return to the beginning of the line” and is equivalent to pressing **RETURN**.

control code: One or more non-printing characters included in a text file whose function is to change the way a printer prints the text. See **control character**.

controller card: A peripheral card that connects a device such as a printer or disk drive to an Apple IIe and controls the operation of the device.

copy-protect: To prevent someone from duplicating the contents of a disk. Compare **write-protect**.

CPU: Abbreviation for *central processing unit*. See **processor**.

current input device: The source, such as the keyboard or a modem, from which a program is currently receiving its input.

current output device: The destination, such as the display screen or a printer, to which a program is currently sending its output.

cursor: A symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter**.

data: Information, especially raw or unprocessed information, used or operated on by a program.

data bits: The computer sends and receives information as a string of bits. These are called *data bits*.

Data Carrier Detect: An RS232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIe) indicating that a communication connection has been established.

Data Communication

Equipment: As defined by the RS232-C standard, any device that transmits or receives information. Usually this is a modem. However, when a modem eliminator is used, the Apple IIe itself looks like a DCE to the other device, and the other device looks like a DCE to the Apple IIe.

data set: A device that performs the modulation/demodulation control functions necessary to provide the compatibility between business machines and communications facilities. See **modem**.

Data Set Ready: An RS232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

Data Terminal Equipment: As defined by the RS232-C standard, any device that generates or absorbs information, thus acting as a terminus of a communication connection.

Data Terminal Ready: An RS232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

DCD: See **Data Carrier Detect**.

DCE: See **Data Communication Equipment**.

debug: To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems. Compare **troubleshoot**.

decimal: The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 through 9.

default: A value, action, or setting that is assumed or set in the absence of explicit instructions otherwise.

deferred execution: The saving of an instruction in a program for execution at a later time as part of a complete program; occurs when the statement is typed with a line number. Compare **immediate execution**.

DELETE: A key on the upper-right corner of the Apple IIe and IIc keyboards that, when pressed, usually erases the character immediately preceding the cursor.

delimiter: A character that is used to mark the beginning or end of a sequence of characters, and which therefore is not considered part of the sequence itself. For example, Applesoft uses the double quotation mark (") as a delimiter for string constants: the string *DOG* consists of the three characters *D*, *O*, and *G*, and does not include the quotation marks. In written English, the space character is used as a delimiter between words.

demodulate: To recover the information being transmitted by a modulated signal; for example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into sound emitted by a speaker.

device: A piece of computer hardware—such as a disk drive, a printer, or a monitor—other than the computer itself. Devices may be built in or peripheral.

device driver: A program that manages the transfer of information between the computer and a peripheral device.

device handler: See **device driver**.

digit: (1) One of the characters 0 through 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 through 9 and A through F in hexadecimal.

digital: Represented in a discrete (noncontinuous) form, such as numerical digits. For example, contemporary digital clocks display the time in numerical form (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog**.

digital data: Data that can be represented by digits—that is, data that are discrete rather than continuously variable. Compare **analog data**.

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package**.

DIP switch: A bank of tiny switches, each of which can be moved manually one way or the other to represent one of two values (usually on and off).

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. The opposite of an **assembler**.

disk: An information-storage medium consisting of a flat, circular, magnetic surface on which information can be recorded in the form of small magnetized spots, in a manner similar to the way sounds are recorded on tape.

disk controller card: A circuit board that provides the connection between one or two disk drives and the Apple IIe.

disk drive: A device that reads information from disks into the memory of the computer and writes information from the memory of the computer onto a disk.

disk envelope: A removable protective paper sleeve used when handling or storing a disk. It must be removed before inserting the disk in a disk drive. Compare **disk jacket**.

diskette: A term sometimes used for the small (5¼-inch), flexible disks on which information is stored.

disk jacket: A permanent protective covering for a disk, usually made of black paper or plastic. The disk is never removed from the jacket, even when inserted in a disk drive. Compare **disk envelope**.

disk operating system: One of several optional software systems for the Apple II family of computers that enables the computer to control and communicate with one or more disk drives.

Disk II drive: One of a number of types of disk drive made and sold by Apple Computer for use with the Apple II family of computers. It uses 5¼-inch flexible (*floppy*) disks.

disk-resident: Stored or held permanently on a disk.

display: *v.* To exhibit information visually. *n.* (1) Information exhibited visually, especially on the screen of a display device, such as a video monitor. (2) A display device.

display color: The color currently being used to draw high-resolution or low-resolution graphics on the display screen.

display device: A device that exhibits information visually, such as a television set or video monitor.

DOS 3.2: An early Apple II operating system. DOS stands for *Disk Operating System*. 3.2 is the version number.

DOS 3.3: One of the operating systems used by the Apple II family of computers. DOS stands for *Disk Operating System*. 3.3 is the version number.

drive: See **disk drive**.

DSR: See **Data Set Ready**.

DTE: See **Data Terminal Equipment**.

DTR: See **Data Terminal Ready**.

dual in-line package: An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side. Often referred to as a **DIP switch**.

Dvorak keyboard: An alternate keyboard layout, also known as the *simplified keyboard*.

effective address: In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

80-column text card: A circuit board that converts the computer's display of text from 40 columns to 80 columns.

80/40 column switch: A switch, either hardware or software, that controls the number of horizontal columns or characters across your screen. A television can display a maximum of 40 characters across, while a video monitor can display 80 characters across the screen.

embedded: Contained within. For example, the string **HUMPTY DUMPTY** is said to contain an embedded space.

emulate: To behave in an identical way. The Apple II 2780/3780 Protocol Emulator and the Apple II 3270 BSC Protocol Emulator, for example, allow your Apple II, II Plus, or IIe, together with the Apple Communications Protocol Card (ACPC), to emulate the operations of IBM 3278 and 3277 terminals and 3274 and 3271 control units.

end-of-command mark: A punctuation mark used to separate commands sent to a peripheral device such as a printer or plotter. Also called a *command terminator*.

end-of-line character: Any character that tells the printer that the preceding text constitutes a full line and may now be printed.

error code: A number or other symbol representing a type of error.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.

Escape character: An ASCII character that allows you to perform special functions when used in combination keypresses.

escape mode: A state of the computer, entered by pressing **[ESC]**, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes, beginning with **[ESC]**, used for positioning the cursor and controlling the display of text on the screen.

even parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits (among the data bits plus the parity bit) an even number.

even/odd parity check: A check that tests whether the number of digits in a group of binary digits is even (even parity check) or odd (odd parity check).

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false. Compare **OR**, **AND**, and **NOT**.

execute: To perform the actions specified by a program command or sequence of commands.

expansion slot: A connector inside the Apple IIe in which a peripheral card can be installed. Sometimes called a *peripheral slot*.

expression: A formula in a program that describes a calculation to be performed.

FIFO: First in, first out.

file: An ordered collection of information stored as a named unit on a peripheral storage medium such as a disk.

firmware: Software stored permanently in hardware: programs in read-only memory (ROM). Such programs (for example, the Applesoft Interpreter and the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory. Compare **hardware**, **software**.

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to

lie at the right end of the number so that the number is interpreted as an integer. Compare **floating-point**.

flag: A variable whose contents (usually 1 or 0, standing for *true* or *false*) indicate whether some condition holds or whether some event has occurred. Used to control the program's actions at some later time.

flexible disk: A disk made of flexible plastic. Often called a *floppy* disk. Compare **rigid disk**.

floating-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to *float* to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

floppy disk: See **flexible disk**.

format: *n.* The form in which information is organized or presented. *v.* (1) To specify or control the format of information. (2) To prepare a blank disk to receive information by dividing its surface into tracks and sectors. Also **initialize**.

form feed: An ASCII character (decimal 12) that causes a printer or other paper-handling device to advance to the top of the next page.

FORTRAN: A contraction of the phrase *FORMula TRANslator*. A widely used, high-level programming language especially suitable for applications requiring extensive numerical calculations, such as in mathematics, engineering, and the sciences. A version called Apple II Fortran is sold by Apple Computer for use with the Apple II Pascal Operating System.

framing error: In serial data transfer, absence of the expected stop bit(s) at the end of a received character.

frequency: The number of complete cycles transmitted per second. Usually expressed in hertz (cycles per second), kilohertz (kilocycles per second), or megahertz (megahertz per second).

full duplex: Capable of simultaneous, two-way communication. Compare *half duplex*.

function: A pre-programmed calculation that can be carried out on request from any point in a program. An instruction that converts data from one form to another.

GAME I/O connector: A special 16-pin connector inside the Apple IIe originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand-control connector**.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's video display screen. Compare **text**.

half duplex: Capable of communication in only one direction at a time. Compare **full duplex**.

hand-control connector: A 9-pin connector on the back panel of the Apple IIe, used for connecting hand controls to the computer. Compare **GAME I/O connector**.

hand controls: Optional peripheral devices, with rotating dial and pushbuttons, that can be connected to the Apple IIe hand control connector. Typically used to control game-playing programs, but can be used in more serious applications as well.

hang: For a program or system to spin its wheels indefinitely, performing no useful work.

hardware: The physical machinery that makes up a computer system. Compare **firmware**, **software**.

hertz: The unit of frequency of vibration or oscillation, also called *cycles per second*. Named for the physicist Heinrich Hertz and abbreviated Hz. The 65C02 microprocessor used in the Apple IIe operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the ten digits 0 through 9 and the six letters A through F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form. Each hexadecimal digit corresponds to a

sequence of four binary digits, or bits. Hexadecimal numbers are preceded by a dollar sign (\$).

high ASCII characters: ASCII characters with decimal values of 128 to 255. Called *high ASCII* because their high bit (first binary digit) is set to 1 (for *on*) rather than 0 (for *off*).

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language. High-level languages available for the Apple IIe include BASIC, Pascal, Logo, and PILOT.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the 65C02 microprocessor, the low-order byte of an address is usually stored first, and the high-order byte second.

high-resolution graphics: The display of graphics on a display screen as a six-color array of points, 280 columns wide and 192 rows high. When the text window is in use, the visible high-resolution graphics display is 280 by 160 points.

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off. Compare **setup time**.

Hz: See **hertz**.

IC: See **integrated circuit**.

immediate execution: The execution of an program instruction as soon as it is typed. Occurs when the line is typed without a line number. This means that you can try out nearly every statement immediately to see how it works. Compare **deferred execution**.

implement: To realize or bring about; for example, a language translator implements a particular language.

IN#: This command designates the source of subsequent input characters. It can be used to designate a device in a slot or a machine-language routine as the source of input.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an

indexed address, contained in an index register and added to the base address to form the effective address.

indexed addressing: A method of specifying memory addresses used in machine-language programming.

index register: A register in a computer processor that holds an index for use in indexed addressing. The 65C02 has two index registers, the **X register** and the **Y register**.

index variable: A variable whose value changes on each pass through a loop. Often called *control variable* or *loop variable*.

infinite loop: A section of a program that will repeat the same sequence of actions indefinitely.

initialize: (1) To set to an initial state or value in preparation for some computation. (2) To prepare a blank disk to receive information by dividing its surface into tracks and sectors. Also **format**.

initialized disk: A disk that is organized into tracks and sectors.

input: Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem.

input/output: Abbreviated **I/O**. The means by which information is transferred between the computer and its peripheral devices.

input routine: A machine-language routine that performs the reading of characters. The standard input routine reads characters from the keyboard. A different input routine might, for example, read them from an external terminal.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number represented inside the computer in fixed-point form. Compare **real number**.

Integer BASIC: A version of the BASIC programming language used by the Apple II family of computers. Integer BASIC is older than Applesoft and capable of processing numbers in integer (fixed-point) form only. Compare **Applesoft BASIC**.

integrated circuit: Networks of microfine wire that conduct electrical impulses. They are etched on silicon wafers and embedded in black plastic.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interface card: A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

interpreter: A language translator that reads a program instruction by instruction and immediately translates each instruction for the computer to carry out. Compare **compiler**.

interrupt: A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

inverse video: The display of text on the computer's display screen in the form of dark dots on a light (or other single phosphor color) background, instead of the usual light dots on a dark background.

I/O: Input/output. The transfer of information into and out of a computer. See **input**, **output**.

I/O device: Input/output device. A device that transfers information into or out of a computer. See **input**, **output**, **peripheral device**.

I/O link: A fixed location that contains the address of an input/output subroutine in the computer's Monitor program.

joystick: An accessory that moves creatures and objects in game programs.

K: Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIe, similar to a typewriter keyboard, used for entering information into the computer.

keyboard input connector: The special connector inside the Apple IIe by which the keyboard is connected to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as **CONTROL-C**) on the keyboard.

keyword: A special word or sequence of characters that identifies a particular type of statement or command, such as *RUN* or *PRINT*.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits. See **K**.

KSW: The symbolic name of the location in the computer's memory where the standard input link is stored. *KSW* stands for *keyboard switch*. See **I/O link**.

language: See **programming language**.

leading zero: A zero occurring at the beginning of a number, deleted by most computing programs.

least significant bit: The right-hand bit of a binary number as written down. Its positional value is 0 or 1.

LIFO: Acronym for *last in, first out*.

line feed: An ASCII character (decimal 10) that ordinarily causes a printer or video display to advance to the next line.

line number: A number identifying a program line in an Applesoft program. Line numbers are necessary for deferred execution.

line width: The number of characters that fit on a line on the screen or on a page.

list: A verb in computer jargon, meaning to display on a monitor, or print on a printer, the contents of the computer memory or a file.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

location: See **memory location**.

logic board: See **main logic board**.

logical operator: An operator, such as AND, that combines logical values to produce a logical result. Compare **arithmetic operator**, **relational operator**.

loop: A section of a program that is executed repeatedly until a limit or condition is met, such as an index variable reaching a specified ending value.

loop variable: See **index variable**.

low-level language: A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIe include 6502 machine language and 6502 assembly language.

low-order byte: The less significant half of a memory address or other two-byte quantity. In the 65C02 microprocessor, the low-order byte of an address is usually stored first, and the high-order byte second.

low-power Schottkey: A type of **TTL** integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.

low-resolution graphics: The display of graphics on a display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high. When the text window is in use, the visible low-resolution graphics display is 40 by 40 blocks.

LS: See **low-power Schottkey**.

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 65C02 microprocessor used in the Apple IIe) has its own form of machine language.

main logic board: A large circuit board that holds RAM, ROM, the microprocessor, custom-integrated circuits, and other components that make the computer a computer.

main memory: The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the computer.

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device can then check for errors by looking for this value on each character.

mask: A pattern of bits for use in bit-level logical operations.

memory: A hardware component of a computer system that can store information for later retrieval. See **main memory**, **random-access memory**, **read-only memory**, **read-write memory**.

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size. In the Apple IIe, a memory location holds one byte, or eight bits, of information.

memory-resident: (1) Stored permanently in main memory as firmware. (2) Held continually in main memory even while not in use. DOS is memory resident.

menu: A list of choices presented by a program, usually on the display screen, from which the user can select.

MHz: Megahertz; one million hertz. See **hertz**.

microcomputer: A computer, such as any of the Apple II family of computers, whose processor is a microprocessor.

microprocessor: A computer processor contained in a single integrated circuit, such as the 65C02 microprocessor used in the Apple IIe.

microsecond: One millionth of a second. Abbreviated μs .

millisecond: One thousandth of a second. Abbreviated ms.

mode: A state of a computer or system that determines its behavior. A manner of operating.

modem: Acronym for *MOdulator/DEModulator*; a peripheral device that enables the computer to transmit and receive information over telephone lines by converting digital signals to analog signals, and vice-versa.

modulate: To modify or alter a signal so as to transmit information. For example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or *AM*) or the frequency (frequency modulation, or *FM*) of a carrier signal.

monitor: See **video monitor**.

Monitor program: A system program built into the firm ware of the Apple IIe, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

most significant bit: The leftmost bit of a binary number as written down. This bit represents 0 or 1 times 2 to the power one less than the total number of bits in the binary number. For example, in the binary number 10000, which contains five digits, the 1 represents 1 times two to the fourth power—or sixteen.

mouse: A small device that you roll around on a flat surface next to your Apple II family system. A small pointer on the screen tracks the movement of the mouse.

nanosecond: One billionth (in British usage, one thousand-millionth) of a second. Abbreviated *ns*.

nested loop: A loop contained within the body of another loop and executed repeatedly during each pass through the containing loop.

nested subroutine call: A call to a subroutine from within the body of another subroutine.

nibble: A unit of information equal to half a byte, or four bits. A nibble can hold any value from 0 to 15. Sometimes spelled *nybble*.

NOT: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true. Compare **AND**, **OR**, **exclusive OR**.

NTSC: (1) Abbreviation for *National Television Standards Committee*. The committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

object code: See **object program**.

object program: The translated form of a program produced by a language translator such as a compiler or assembler. Also called *object code*. Compare **source program**.

odd parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number.

opcode: See **operation code**.

operand: A value to which an operator is applied. The value on which an opcode operates.

operating system: The most fundamental program in a computer. It organizes the actions of the various parts of the computer and allows it to use other programs.

operation code: The part of a machine-language instruction that specifies the operation to be performed. Often called *opcode*.

operator: A symbol or sequence of characters, such as + or *AND*, specifying an operation to be performed on one or more values (the operands) to produce a result. See **arithmetic operator**, **relational operator**, **logical operator**, **unary operator**, **binary operator**.

option: An **argument** that is optional.

OR: A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false. Compare **exclusive OR**, **AND**, **NOT**.

output: Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

output routine: A machine-language routine that performs the sending of characters. The standard output routine writes characters to the screen. A different output routine might, for example, send them to a printer.

overflow: The condition that exists when an attempt is made to put more data into a memory area than it can hold.

override: To modify or cancel a long-standing instruction with a temporary one.

overrun: A condition that occurs when the processor does not retrieve a received character from the receive data register of the **ACIA** before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

page: (1) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A screenful of information on a video display. With the Apple IIe, a page consists of 24 lines of 40 or 80 characters each.

page zero: See **zero page**.

parallel interface: An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

parity: Maintenance of a sameness of level or count, usually the count of 1 bit in each character, for error checking.

Pascal: A high-level programming language with statements that resemble English sentences. Pascal was designed to teach programming as a systematic approach to problem solving. Named after the philosopher and mathematician, Blaise Pascal.

pass: A single execution of a loop.

PC board: See **printed-circuit board**.

peek: To read information directly from a location in the computer's memory.

peripheral: At or outside the boundaries of the computer itself, either physically (as a peripheral device) or in a logical sense (as a peripheral card).

peripheral bus: The bus used for transmitting information between the computer and peripheral devices connected to the computer's expansion slots.

peripheral card: A removable printed circuit board that plugs into one of the expansion slots in the Apple IIe. It expands or modifies the computer's capabilities by connecting a peripheral device or performing some subsidiary or peripheral function.

peripheral device: An auxiliary piece of equipment—such as a video monitor, disk drive, printer, or modem—used in conjunction with a computer and under the computer's control. Often (but not necessarily)

physically separate from the computer and connected to it by wires, cables, or some other form of interface, typically by means of a peripheral card.

peripheral slot: See **expansion slot**.

phase: (1) A stage in a periodic process. A point in a cycle. For example, the 65C02 micro processor uses a clock cycle consisting of two phases called $\phi 0$ and $\phi 1$. (2) The relationship between two periodic signals or processes. For example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

PILOT: Acronym for *Programmed Inquiry, Learning, Or Teaching*. A high-level programming language designed to enable teachers to create computer-aided instruction (CAI) lessons that include color graphics, sound effects, lesson text, and answer checking. A version called Apple II PILOT is sold by Apple Computer for use with the Apple II family of computers.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All else being equal, processors that have this feature run faster than those without it.

plotting vector: A code representing a single step in drawing a shape on the high-resolution graphics screen, specifying whether to plot a point at the current screen position and in what direction to move (up, down, left, or right) before processing the next vector.

point of call: The point in a program from which a subroutine or function is called.

pointer: An item of information consisting of the memory address of some other item. For example, Applesoft maintains internal pointers to (among other things) the most recently stored variable, the most recently typed program line, and the most recently read data item.

poke: To store information directly into a location in the computer's memory.

pop: To remove the top entry from a stack.

power supply: A box that draws electrical power from a power outlet and converts it to the power the computer can use to do its computing.

power supply case: The metal case inside the Apple IIe that houses the power supply.

PR#: The PR# command sends output to a slot or a machine-language program. It specifies an output routine in the ROM on a peripheral card or in a machine-language routine in RAM by changing the address of the standard output routine used by the computer.

precedence: The order in which operators are applied in evaluating an expression.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, to which integrated circuits and other electronic components are connected.

procedure: In the Pascal programming language, a set of instructions that work as a unit; equivalent to the subprogram in BASIC.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

ProDOS: An Apple II operating system designed to support mass storage devices like the ProFile as well as flexible disk storage devices. ProDOS stands for *Professional Disk Operating System*.

ProDOS command: Any one of the 28 commands recognized by ProDOS. Each has its own syntax, all can be used within programs, and all but five (text file commands) can be used from immediate mode.

program: *n.* A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. In Applesoft, a sequence of program lines, each with a different line number. *v.* To write a program.

programmer: The author of a program; one who writes programs.

programming: The activity of writing programs.

programming language: A set of rules or conventions for writing programs.

prompt: *n.* A message on the screen. *v.* To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character: A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character] is used by the Applesoft BASIC interpreter, > by Integer BASIC, and * by the system Monitor program. Also called *prompting* character.

prompt line: A message displayed on the screen to prompt the user for some action. Also called *prompting message*.

protocol: A set of rules for sending and receiving data on a communications line.

push: To add an entry to the top of a stack.

queue: A list in which entries are added at one end and removed at the other, causing entries to be removed in FIFO (first-in first-out) order. Compare **stack**.

radio-frequency modulator: A device that transforms your television set into a computer display device.

RAM: See **random-access memory**.

random-access memory (RAM): Memory in which the contents of individual locations can be referred to in an arbitrary or random order; the readable and writable memory of the Apple IIe. Its contents are usually filled with programs from a disk, and they are lost when the Apple IIe is turned off. This term is often used misleadingly to refer to read-write memory, but, strictly speaking, both read-only and read-write memory can be accessed in random order. Random-access means that each unit of storage has a unique address and a method by which each unit can be immediately read from or written to. Compare **read-only memory**, **read-write memory**.

random-access text file: A text file that is partitioned into an unlimited number of uniform-length compartments called records. When you open a random-access text file for the first time, you must specify its record length. No record is placed in the file until written to. Each record can be individually read from or written to—hence, *random-access*.

raster: The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive dots on the individual lines of the raster.

read: To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory (ROM): Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and

can never be erased or changed. Compare **random-access memory**, **read-write memory**.

read-write memory: Memory whose contents can be both read and written; often misleadingly called random-access memory, or RAM. The information contained in read-write memory is erased when the computer's power is turned off, and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare **random-access memory**, **read-only memory**.

real number: A number that may include a fractional part; represented inside the computer in floating-point form. Compare **integer**.

register: A location in a computer processor where an item of information is held and modified under program control.

relational operator: An operator, such as $>$, that compares numeric values to produce a logical result. Compare **arithmetic operator**, **logical operator**.

reserved word: A word or sequence of characters reserved by a programming language for some special use, and therefore unavailable as a variable name in a program.

resident: See **memory-resident**, **disk-resident**.

return address: The point in a program to which control returns on completion of a subroutine or function.

RF modulator: See **radio-frequency modulator**.

ROM: See **read-only memory**.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

row: A horizontal arrangement of character spaces or graphics points on the screen.

RS232 cable: Any cable that is wired in accordance with the RS232 standard, which is the common data communications interface standard.

run: (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **window**.

serial interface: An interface in which information is transmitted sequentially, one bit at a time, over a single wire or channel. Compare **parallel interface**.

setup time: The amount of time a signal must be valid in advance of some event. Compare **hold time**.

silicon: A non-metallic, semiconducting chemical element from which integrated circuits are made. Not to be confused with silica—that is, silicon dioxide, such as quartz, opal, or sand—or with silicone, any of a group of organic compounds containing silicon.

simple variable: A variable that is not an element of an array.

simplified keyboard: The Dvorak keyboard.

6502: The type of microprocessor used in the Apple II, II Plus, and original IIe.

65C02: The type of microprocessor used in the enhanced Apple IIe and the Apple IIc.

slot: A narrow socket inside the computer where you can install peripheral device cards.

soft switch: A means of changing some feature of the computer from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

software: Instructions that tell the computer what to do. They're usually stored on disks. Compare

hardware, firmware.

source program: The original form of a program given to a language translator such as a compiler or assembler for conversion into another form; sometimes called *source code*. Compare **object program**.

space character: A text character whose printed representation is a blank space, typed by pressing the `[SPACE]` bar.

stack: A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in first-out) order. Compare **queue**.

standard instruction: An instruction automatically present when no superseding instruction has been received.

start up: To get the system running. For example, In the context of ProDOS, starting up is the process of reading the ProDOS program (in the files PRODOS and BASIC.SYSTEM) from the disk, and running it.

starting value: The value assigned to the index variable on the first pass through a loop.

startup disk: A disk containing an operating system and a self-starting program.

statement: A unit of a program in a high-level language that specifies an action for the computer to perform, typically corresponding to several instructions of machine language.

step value: The amount by which the index variable changes on each pass through a loop.

string: An item of information consisting of a sequence of text characters.

stroke: A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

synchronous transmission: A transmission process that requires an integral number of unit (time) intervals between any two significant instances. In synchronous communications, the transmitter and receiver are in step with each other, and characters being transmitted follow one after the other at regular intervals. Compare **asynchronous transmission**.

syntax: The rules governing the structure of statements or instructions in a programming language; a representation of a command that specifies all the possible forms the command can take.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose.

system configuration: See **configuration**.

system program: A program that makes the resources and capabilities of the computer available for general purposes, such as an operating system or a language translator. Compare **application program**.

system software: The component of a computer system consisting of system programs.

TAB: An ASCII character that commands a device such as a printer to start printing at a preset location (called a tab stop). There are two such characters; horizontal tab (hex \$09) and vertical tab (hex \$0B).

television set: A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIe. Compare **video monitor**.

terminal: A device consisting of a typewriter-like keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as those in the Apple II family of computers typically have all or part of a terminal built into them.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on a display screen. Compare **graphics**.

text window: An area on the video display screen within which text is displayed and scrolled.

traces: Electrical roads that connect the components on a circuit board.

transistor-transistor logic (TTL): (1) A type of integrated circuit used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical zeros and ones.

troubleshoot: To locate and correct the cause of a problem or malfunction in a computer system. Typically used to refer to hardware-related problems. Compare **debug**.

TTL: See **transistor-transistor logic**.

turnkey disk: A disk that executes a specific application program when you use that disk to start the computer.

turnkey program: A program, such as a game or application, that runs automatically when the disk that the program is on is used to start up the computer.

unary operator: An operator that applies to a single operand; for example, the minus sign (-) in a negative number such as -6 is a unary arithmetic operator. Compare **binary operator**.

unconditional branch: A branch that does not depend on the truth of any condition. Compare **conditional branch**.

value: An item of information that can be stored in a variable, such as a number or a string.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location. Compare **constant**.

vector: (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device. Compare **television receiver**.

volume: A general term referring to a storage device; a source or destination of information. A volume has a name and a volume directory with the same name. Its information is organized into files.

window: The portion of a collection of information (such as a document, picture, or worksheet) that is visible on the display screen.

word: A group of bits of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

write-enable notch: The square cutout on one edge of a disk's jacket that permits information to be written on the disk. If there is no write-enable notch, or if it is covered with a write-protect tab, information can be read from the disk but not written onto it.

write-protect: To protect the information on a disk by covering the write-enable notch with a write-protect tab, preventing any new information from being written onto the disk. Compare **copy protect**.

write-protect tab: A small adhesive sticker used to write-protect a disk by covering the write-enable notch.

X register: One of the index registers in the 65C02 microprocessor.

Y register: One of the index registers in the 65C02 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple IIe, also called page zero. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.

Bibliography

Addendum to the Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1984.

Apple II Monitors Peeled. Cupertino, Calif.: Apple Computer, Inc., 1978.

Currently not updated for Apple IIe and IIfx, but a good introduction to Apple II series input/output procedures; also useful for historical background.

Apple IIe Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1982.

Applesoft BASIC Programmer's Reference Manual, Volumes 1 and 2. For the Apple II, IIe, and IIfx. Reading, Mass.: Addison-Wesley, 1982, 1985. ISBN 0-201-17722-6.

Applesoft Tutorial. Reading, Mass.: Addison-Wesley, 1983, 1985. ISBN 0-201-17724-2.

"Characteristics of Television Systems." *C.C.I.R. Report*, Rep. 624 (1970-1974), pp. 22-52.

"Colorimetric Standards in Colour Television." *C.C.I.R. Report*, Rep. 476-1 (1970-1974), pp. 21-22.

Leventhal, Lance. *6502 Assembly Language Programming*. Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

Sims, H. V. *Principles of PAL Colour Television and Related Systems*. London, England: Newnes-Butterworth, 1969. ISBN-0-592-05970-7.

Synertek Hardware manual. Santa Clara, Calif.: Synertek Incorporated, 1976.

Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500 series microcomputers.

Synertek Programming manual. Santa Clara, Calif.: Synertek, Incorporated, 1976.

The only currently available manufacturer's programming manual for 6500 series microcomputers.

"Video-Frequency Characteristics of a Television System to Be Used for the International Exchange of Programmes Between Countries That Have Adopted 625-Line Colour or Monochrome Systems." *C.C.I.R.*, Recommendation 472-1 (1970-1971), pp. 53-54.

Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte* Vol. 5, No. 11 (November, 1980).

----- "A Simplified Theory of Video Graphics, Part II." *Byte* Vol. 5, No. 12 (December, 1980).

----- "More Colors for Your Apple." *Byte* Vol. 4, No. 6 (June, 1979).

----- "True Sixteen-Color Hi-Res." *Apple Orchard* Vol. 5, No. 1 (January, 1984).

Wozniak, Steve: "System Description: The Apple II." *Byte* Vol. 2, No. 5 (May, 1977).

----- "SWEET16: The 6502 Dream Machine." *Byte* Vol. 2, No. 10 (October, 1977).

Index

Cast of Characters

* (asterisk) as prompt character 60
^ (caret) 122, 125
: (colon) as Monitor command 103
> (greater than sign) as prompt character 60
☐ 61
☐ 11, 13, 226
.(period) as Monitor command 100
φ0 (phi 0) 162-164, 170, 171, 180-181
φ1 (phi 1) 162-164, 170, 171, 180-181
φ2 (phi 2) 162
? (question mark) prompt character 60
☐ 62
](right bracket) as prompt character 60
☐ 11, 13, 226
14M signal 163
40-column text 20-21
 display pages 27
 generation 179
 memory map 32, 177
 with TV set 16
6502 microprocessor 5, 6
 differences from 65C02 206-207
65C02 microprocessor xxix, 5, 6, 206-216
 data sheet 208-216
 differences from 6502 6, 206-207
 specifications 161-164
 timing 162-164
65C02 stack 75
80COL soft switch 29
80-column firmware xxx, 49-50
 activating 49
 control characters with 273-275

80-column text 20-21
 differences in Apple II family 227
 display pages 27
 generation 179
 map 33
 signals 197
 with Applesoft xxx
 with Pascal xxx
 with TV set 16
80-Column Text Card 84, 132, 149, 268-275
80STORE soft switch 29, 31, 84, 86, 87, 197

A

A register 146
A1 89
A2 89
A4 89
accumulator 136, 148
ACIA 289
address bus 162
address transformation 176
addressing
 display pages 30-36, 175-178
 I/O locations 136-137
 RAM 138, 170-173
 ROM 169
addressing, indirect 75
addressing, relative 119, 125, 135
ALTCHAR soft switch 29
alternate character set 19-20, 226
 on original IIe 20
ALTZP soft switch 82, 87, 89
analog inputs 42, 43
animation 229
annunciators 40, 43
any-key-down flag 12
Apple keys 11, 13
 differences in Apple II family 226
Applesoft BASIC xxx, 12, 103, 233
 and lowercase xxxi
 and uppercase 48
 80-column support xxx
 tabbing with original Apple IIe 271-272
 use of zero page 77
Apple II compatibility with Apple IIe 48-50
Apple II family differences 226-230
Apple IIc interrupt differences 156
Apple IIe, differences between original and enhanced xxix-xxxii
 ASCII input mode 105
 COUT1 subroutine 54
 interrupt support 130, 148
 microprocessor 6
 Mini-Assembler 121
 Monitor Search command 108
 MouseText 16, 20
 slot 3 143
 tabbing in Applesoft 271-272
 using **CAPS LOCK** 48
Apple IIe 80-Column Text Card 84, 132, 268-275
Apple IIe Extended 80-Column Text Card 84, 132, 268-275
arithmetic, hexadecimal 114
arrow keys 61, 62
ASCII codes 14-15
ASCII input mode 104-105
assemblers 119
assembly language 233
asterisk (*) as prompt character 60
auxiliary firmware 84-91

- auxiliary memory 84-91
 - differences in Apple II family 228
 - map 85
 - moving data to 89
 - soft switches 87
 - subroutines 88
- auxiliary RAM 84
- auxiliary slot 7, 49
 - differences in Apple II family 228
 - signals 197-199
- AUXMOVE subroutine 88, 89, 143

B

- backspacing 61
- bank-switched memory 79-83, 85, 227
 - map 80
- bank switches 80-83, 85
 - reading 83
- BASIC, Applesoft xxx, 12, 103, 233
 - and lowercase xxxi
 - and uppercase 48
 - 80-column support xxx
 - tabbing with original Apple IIe 271-272
 - use of page 3 76
 - use of zero page 77
- BASIC, Integer 12, 233
 - and bank-switched memory 79
 - and reset 81
 - and uppercase 48
 - use of page 3 76
 - use of zero page 78
- BASICIN subroutine 57, 218
 - address in I/O link 51
- BASIC Monitor command 112
- BASICOUT subroutine 63, 218
 - address in I/O link 51

- baud rate for SSC 280
- BEL character 52
- BELL subroutine 218
- BELL1 subroutine 38, 218
- bit definition 236
- bit mapping of graphics 24-26
- booting 268-269
- break instructions 155
- BRK handler 155
- BRK instruction 155
- BRK vector 147
- BS character 52
- byte definition 237

C

- canceling lines 61
- CAN character 53
- CAPS LOCK** 11
 - for older software compatibility 48
- caret (^) 122, 125
- carriage returns with SSC 283
- cassette I/O 38-39, 189
 - commands 109-111
 - soft switches 38
- central processing unit (CPU) 6
 - See also* 65C02 microprocessor
- CH 51
- changing memory contents 103-108
- character code 12
- character generator ROM 179
- character sets, text 19-20
 - differences among Apple II models 226-227
- CHARGEN signal 185
- circuit board 4-5
 - connectors 6
- clear-strobe switch 12
- CLEOLZ subroutine 49, 64, 219
- clock rate 161
- clock signals 162
- CLREOL subroutine 49, 63, 218
- CLREOP subroutine 49, 64, 219
- CLRSCR subroutine 64, 219
- CLRTOP subroutine 64, 219
- cold-start reset 92
- colon (:) as Monitor command 103
- color graphics with black-and-white monitors 16
- colors
 - double-high-resolution graphics 26
 - high-resolution graphics 24-25, 183
 - low-resolution graphics 23
- command characters, Monitor 99
- comma tabbing with original Apple IIe 271-272
- complementary decimal values 12
- connectors
 - back panel 8
 - cassette I/O 8, 38
 - D-type 8
 - game I/O 7, 13
 - hand control 8, 39-42
 - 9-pin 8, 39
 - phone jacks 8, 38
 - power 161
 - RCA-type jack 8
 - video monitor 8, 186
- control characters 245, 249
 - with BASICOUT 52-53
 - with COUT1 52
 - with 80-column firmware 273-275
 - with Pascal I/O protocol 68-69
- Control-U 50
- CONTROL** 11
- CONTROL** + **B** Monitor command 112

- CONTROL**-**E** Monitor command 109
- CONTROL**-**K** Monitor command 113
- CONTROL**-**P** Monitor command 113
- CONTROL**-**X** 61
- CONTROL**-**Y** Monitor command 117
- COUT subroutine 51, 64, 219
 - deactivating 80-column firmware 50
- COUT1 subroutine 51, 64, 134, 219
 - address in I/O link 50
 - on original Apple IIe 54
- cover 2
- CP/M 233
 - starting up with 268
- CPU 6
 - See also* 65C02 microprocessor
- CR character 53
- CROUT subroutine 64, 219
- CROUT1 subroutine 65, 219
- CSW link 139
- current, supply 159
- cursor-control keys 11
- cursor motion in escape mode 58-59
- cursor position 51-57
- custom IC's 164-168
- CV 51
- cycle stealing 170

D

- D-type connector 8
- daisy chains, interrupt and DMA
 - 193-194, 203
- data bus 162
- data format for SSC 281
- DC1 character 53
- DC2 character 53
- DC3 character 53

- decimal values 12
 - converting to hexadecimal 238-239
 - negative 240-241
- device assignment, peripheral card
 - 144
- device identification 144
- DEVICE SELECT' signal 131
- DHIRES soft switch 29
- Diagnostics ROM 169
- differences among Apple II models
 - 226-230
- differences between original and
 - enhanced Apple IIe xxix-xxxi
- ASCII input mode 105
- COUT1 subroutine 54
- interrupt support 130, 148
- microprocessor 6
- Mini-Assembler 121
- Monitor Search command 108
- MouseText 16, 20
- slot 3 143
- tabbing in Applesoft 271-272
- using **CAPS LOCK** 48
- disassemblers 119
- display, video 16-36
 - address transformation 176-177
 - double-high-resolution graphics 185
 - 80-column text 179
 - formats 17, 56
 - 40-column text 179
 - generation 173-185, 230
 - high-resolution graphics 183-184
 - low-resolution graphics 182-183
 - memory addressing 175-178
 - modes 17, 20-26, 28-30, 179-185
 - pages 26-28, 30-36, 76
 - refreshing 170
 - specifications 17
 - text 179-181

- DMA daisy chain 193-194, 203
- DOS 3.3 xxix, 140, 232
 - and uppercase 48
 - starting up with 269
 - use of page 3 76
 - use of zero-page 78
- double-high-resolution graphics 17, 18,
 - 25-26
 - colors 26
 - display pages 27
 - generation 185
 - map 36
 - memory pages 25
- double-high-resolution Page 1 76

E

- editing with GETLN 61
- 80COL soft switch 29
- 80-column firmware xxx, 49-50
 - activating 49
 - control characters with 273-275
- 80-column text 20-21
 - differences in Apple II family 227
 - display pages 27
 - generation 179
 - map 33
 - signals 197
 - with Applesoft xxx
 - with Pascal xxx
 - with TV set 16
- 80-Column Text Card 84, 132, 149,
 - 268-275
- 80STORE soft switch 29, 31, 84, 86, 87,
 - 197
- EM character 53
- EN80' signal 197

enhanced Apple IIe *See* differences
 between original and enhanced
 Apple IIe
 ENKBD' signal 187
 entry points for I/O routines 145-146
 escape codes 58-59
 escape mode 58-59
 ESC character 53
 ETB character 53
 EXAMINE command 108-109
 examining memory 100
 expansion ROM space 132-134
 expansion slot 3 49
 expansion slots 6-7, 130-143
 signals 192-196
 Extended 80-Column Text Card 84,
 132, 268-275

F

FF character 52
 firmware
 auxiliary 84-91
 80-column xxx, 49-50
 I/O 46-69
 Monitor subroutines 46-69
 Pascal 1.1 protocol 67-69, 144-146
 slot 3 67
 flag, any-key-down 12
 FLASH command 270-271
 flashing format 18-19, 56-57
 forced cold-start reset 93
 FORTRAN 234
 40-column text 20-21
 display pages 27
 generation 179
 memory map 32, 177
 with TV set 16
 FS character 53

G

game I/O
 connectors 13
 signals 190-191
 GET command 269
 GETLN subroutine 57, 60-62, 220
 editing with 61
 input buffer 76
 line length 61
 used by Monitor 99
 with 80-column card 269
 GETLN1 subroutine 220
 GETLNZ subroutine 220
 GO command 118
 graphics, double-high-resolution 17,
 18, 25-26
 colors 26
 display pages 27
 generation 185
 map 36
 memory pages 25
 graphics, high-resolution 17, 18, 23-25,
 addressing display pages 31, 35
 bit patterns 242-243
 colors 24-25, 183
 display pages 23, 27
 generation 183-184
 map 35
 graphics, low-resolution 17, 18, 22-23,
 colors 23
 display pages 27
 generation 182-183
 map 34
 with TV set 16
 graphics modes 22-26
 bit-mapping 24-26
 greater than sign (>) as prompt
 character 60
 GS character 53

H

hand control connectors 8, 39-42
 hard disk with Pascal xxxi
 hexadecimal arithmetic 114
 hexadecimal values 12
 converting to decimal 238-239
 converting to negative decimal
 240-241
 high-resolution graphics 17, 18, 23-25
 addressing display pages 31, 35
 bit patterns 242-243
 colors 24-25, 183
 display pages 27
 generation 183-184
 map 35
 high-resolution Page 1 23, 27, 76
 high-resolution Page 2 23, 76
 HIRES soft switch 29, 86, 87
 HLINE subroutine 65, 220
 HOME command 270-271
 HOME subroutine 49, 65, 220
 HTAB command xxx
 with original Apple IIe 272
 humidity, operating 158

I, J

I/O
 addressing 136-137
 circuits 187-191
 devices, built-in 10-42
 entry points 145-146
 firmware, built-in 46-69
 links 50-51, 76, 139-140
 memory for peripheral cards
 130-131
 memory map 141
 Pascal protocol 67-69, 143, 144-146
 switching memory 141-142

I/O SELECT' signal 131-132
 identification byte xxix, 230
 IN# command 113
 index register 136
 indirect addressing 75
 input buffer 76
 INPUT command 269
 input devices *See* I/O devices
 input/output *See* I/O
 Input/Output Unit (IOU) 5, 6, 166-167, 187
 inputs
 analog 37, 42
 hand control 37
 secondary 37-42, 43
 switch 37-41
 See also I/O devices
 INT IN pin 147
 INT OUT pin 147
 Integer BASIC 12, 233
 and bank-switched memory 79
 and reset 81
 and uppercase 48
 use of page 3 70
 use of zero page 78
 interpreter ROM 5
 interrupt handler
 built-in 146, 149-150
 user's 154
 interrupts xxx, 146-156
 and card in auxiliary slot 49
 daisy chain 193-194, 203
 definition 147
 original Apple IIe differences 148
 priority 147
 sequence 151
 interrupt vector 150
 INVERSE command 270-271
 inverse display format 18-19, 56-57, 112

IOREST subroutine 220
 IOSAVE subroutine 220
 IOU (Input/Output Unit) 5, 6, 166-167, 187
 IOUDIS soft switch 29
 IRQ vector 147
 IRQ' signal 147

K

KBD' signal 187
 keyboard 3, 10-15
 automatic repeat function 10
 circuits 187-188
 differences in Apple II family 226
 memory locations 12
 rollover 10
 specifications 11
 KEYBOARD command 113
 keyboard encoder 5, 12
 keyboard ROM 5
 keyboard strobe 13
 KEYIN subroutine 57, 58-59, 221
 address in I/O link 50
 keypad 188
 keys and ASCII codes 14-15
 KSW link 139

L

language card 83
 differences in Apple II family 227
 LED 3
 ⌘ key 61
 LF character 52
 line feeds with SSC 284
 links, I/O 50-51
 address storage 70
 changing 139-140

LIST command 119
 low-resolution graphics 17, 18, 22-23
 colors 23
 display pages 27
 generation 182-183
 map 34
 with TV set 16

M



machine language 118-120
 mapping display addresses 176-177
 maps *See* memory maps
 memory
 addressing 168
 auxiliary 84-91
 bank-switched 79-83, 86-88, 227
 changing contents 103-108
 display 175-178
 examining 100
 filling 115-116
 for peripheral cards 130-135
 I/O space 141-142
 organization 72-95
 sharing 88
 text window locations 55
 used by SSC 289
 memory dump 100-102
 Memory Management Unit (MMU) 5, 6, 164-165
 memory maps
 auxiliary memory 85
 bank-switched areas 80
 double-high-resolution graphics 36
 80-column text 33
 40-column text 32, 177
 high-resolution graphics 35
 I/O 141
 low-resolution graphics 34
 main memory 73
 RAM 74

memory pages, reserved 75-79
 microprocessor *See* 65C02 and 6502
 microprocessors
 Mini-Assembler 121-125
 errors 122
 instruction formats 124-125
 starting 121
 MIXED soft switch 29
 Monitor, System 98-128
 command summary 125-128
 command syntax 99
 creating commands 117
 enhancements xxvi
 firmware subroutines 46-69
 returning to BASIC 112
 ROM listings 294-375
 use of page 3 76
 use of zero page 77
 Monitor ROM 169
 listings 294-375
 MouseText characters 17, 19, 247
 MOVE command 105, 115
 MOVE subroutine 221
 MSLOT 150, 154

N

NAK character 53
 negative decimal values 12
 converting 240
 NEXTCOL subroutine 221
 9-pin connectors 8, 39
 NORMAL command 270-271
 normal format 18-19, 112
 NTSC standard 16, 24, 173

O

 11, 13, 226
 operating systems 232-233
 original Apple IIe
 ASCII input mode 105
 COUT1 subroutine 54
 interrupt support 130, 148
 microprocessor 6
 Mini-Assembler 121
 Monitor Search command 108
 MouseText 16, 20
 slot 3 143
 startup display 6
 tabbing in Applesoft 271-272
 using  48
 output *See* I/O
 overheating 158

P

Page 1, double-high-resolution 76
 Page 1, high-resolution 23, 27, 76
 Page 1, text 27
 Page 2, high-resolution 23, 76
 Page 2, text 27
 page 3 vectors 94
 page zero *See* zero page
 PAGE2 soft switch 29, 31, 84, 86, 87
 pages, reserved memory 75-79
 PAL device 168
 parity for SSC 281-282
 Pascal xxix, 234, 275
 and bank-switched memory 79
 I/O subroutines 46
 starting up with 268
 Pascal 1.1 firmware protocol 67-69,
 143, 144-146
 Pascal operating system 232

period (.) as Monitor command 100
 peripheral address bus 192-193, 194
 peripheral cards
 device assignment 144
 I/O memory space 130-131, 141
 programming for 130-156
 RAM space 134-135
 ROM space 131-132
 peripheral data bus 193
 differences in Apple II family 230
 peripheral slots *See* expansion slots
 $\phi 0$ (phi 0) 162, 164, 170, 171, 180-181
 $\phi 1$ (phi 1) 162, 164, 170, 171, 180-181
 $\phi 2$ (phi 2) 162
 phone jacks 8, 38
 PINIT subroutine 67
 pipelining 161
 PLOT subroutine 65, 221
 POKE command 272
 power connector 161
 power supply 4, 159-160
 PR# command 113
 PRBL2 subroutine 65, 221
 PRBLNK subroutine 221
 PRBYTE subroutine 65, 221
 PREAD subroutine 42, 67, 222
 PRERR subroutine 65, 222
 PRHEX subroutine 66, 222
 primary character set 19-20
 PRINTER command 113
 PRNTAX subroutine 66, 222
 ProDOS 103, 140, 232
 interrupt support 148
 starting up with 269
 use of page 3 76
 use of zero-page 79


ProFile hard disk xxxi
Programmed Array Logic (PAL) device
 5, 168
prompt characters 60
PSTATUS subroutine 69
PWRITE subroutine 68

Q

Q3 signal 163
question mark (?) prompt character 60

R


R/W80 signal 197
radio frequency modulator 7
RAM
 addressing 138, 170-173
 allocation 74-79
 auxiliary 84
 space for peripheral cards 134-135
 timing signals 171-173
RAMRD soft switch 86, 87
RAMWR soft switch 86, 87
random number generator 58
RDALTCHAR soft switch 29
RDALTZP soft switch 82
RDBNK2 soft switch 82
RDCHAR subroutine 222
RDDHIRES soft switch 29
RD80COL soft switch 29
RD80STORE soft switch 29
RDHIRES soft switch 29
RDIODIS soft switch 29
RDKEY subroutine 47, 57, 60, 222, 269
RDLGRAM soft switch 82
RDMIXED soft switch 29
RDPAGE2 soft switch 29
RDTEXT soft switch 29
READ subroutine 39, 222
READ tape command 110-111

refreshing the display 170
registers 146, 162
 A register 146
 accumulator 136, 148
 examining and changing 108-109
 index 136
 X register 146
 Y register 146
relative addressing 119, 125, 135
reserved memory pages 75-79
RESET 11, 13, 226
reset routine 91-95
 and bank switches 81
 differences in Apple II family 229
reset vector 93-94
RETURN Monitor command 125
retype function 62
RF modulator 7
RGB-type monitor 185
 62
right bracket (]) as prompt character
 60
rollover, N-key 10
ROM
 addressing 169
 expansion 132-134
 interpreter 5
 keyboard 5
 Monitor listings 294-375
 space for peripheral cards 131-132
 video 5
ROMEN1 signal 169
ROMEN2 signal 169

S

schematic diagram 200-203
SCRN subroutine 66, 223
SEARCH command 108

self-test 13, 95
 differences in Apple II family 229
SETCOL subroutine 66, 223
SETINV subroutine 223
SETNORM subroutine 223
SHIFT 11
shift-key mod 41
short circuits 160
SI character 53
signals
 auxiliary slot 197-199
 expansion slot 192-196
 game I/O connector 191
 IOU 167
 keyboard connector 188
 keypad connector 188
 MMU 165
 PAL device 168
 RAM timing 172-173
 65C02 timing 163
 speaker connector 189
 video connector 186
 video timing 180-181, 184
signature byte 230
6502 microprocessor 6
 differences from 65C02 6, 206-207
65C02 microprocessor xxix, 5, 6,
 206-216
 comparison with 6502 6, 206-207
 data sheet 208-216
 specifications 161-164
 timing 162-164
slot, auxiliary 7, 49
slot number, finding 136
slot 3 49, 149
 firmware 67
 in original Apple IIe 143
slots, expansion 6-7, 130-143
 signals 192-196

SLOT3ROM soft switch 49, 142
 SLOTXROM soft switch 142
 SO character 53
 soft switches
 auxiliary memory 84, 87
 bank switches 80-83, 85
 differences in Apple II family 228
 display 28-30
 for bank switching 83,85
 I/O memory 141-142
 implemented by IOU 166
 implemented by MMU 164
 speaker 38
 11, 13, 226
 SPC command xxx
 speaker 3, 37, 189
 connector 189
 soft switch 38
 specifications, environmental 158
 stack pointers 75, 152
 stack
 auxiliary 152-153
 main 152-153
 65C02 75
 standard I/O links 50-51
 address storage 76
 changing 139-140
 starting up 268-269
 startup display 6
 startup drives xxix-xxx
 stop-list feature 54
 strobe bit 13
 strobe output 40, 43
 STSBYTE 287
 SUB character 53

subroutines
 directory of 218-224
 output 62-66
 Pascal I/O protocol 67-69
 standard I/O 46-69
 See also names of subroutines
 Super Serial Card 278-293
 command character 280
 commands 280-287
 error codes 287-288
 memory use 289-292
 scratchpad RAM 292
 terminal mode 286-287
 switch 0 41, 43
 switch 1 41, 43
 switches *See* soft switches
 switch inputs 41, 43
 SYN character 53
 System Monitor *See* Monitor, System

T

tabbing
 TAB command xxx
 with original Apple IIe 271-272
 television set 16
 temperature
 case 159
 operating 158
 text cards 84, 132, 149, 268-275
 text character sets
 alternate 19-20
 primary 19-20, 226, 227
 text display 18-21, 179-181
 flashing format 56-57
 inverse format 18, 56-57
 normal format 18
 See also 40-column text and
 80-column text

text Page 1 27, 76
 text Page 2 27, 76
 TEXT soft switch 29
 text window 54-55
 memory locations 55
 timing signals
 expansion slots 194
 RAM 172-173
 65C02 microprocessor 162-164
 video 180-181, 184

U

US character 53
 user's interrupt handler 154

V

vectors
 BRK 147
 interrupt 150
 IRQ 147
 page 3 94
 reset 93-94
 VERIFY command 107, 116
 VERIFY subroutine 223
 vertical sync 229
 VID7M signal 163
 video counters 174-175
 video display *See* display, video
 video display pages 23, 25, 26-28
 video monitor 16
 connector 8, 186
 video output signals 186
 video ROM 5

video standards 173
VLINE subroutine 66, 224
voltage
 line 158
 supply 159
VT character 52
VTABZ subroutine 66

W

WAIT subroutine 224
warm-start reset 92
WRITE subroutine 38, 224
WRITE tape command 109-110

X

XFER subroutine 88, 90, 143, 153
X register 146

Y

Y register 146

Z

zero page 75, 77-79

